


Please cite the Published Version

Asiamah, Emmanuel Acheampong, Akrasi-Mensah, Nana Kwadwo, Odame, Prince, Keelson, Eliel, Agbemenu, Andrew Selasi, Tchao, Eric Tutu, Al-Khalidi, Mohammed  and Klogo, Griffith Selorm (2025) A storage-efficient learned indexing for blockchain systems using a sliding window search enhanced online gradient descent. The Journal of Supercomputing, 81 (1). 321 ISSN 0920-8542

DOI: <https://doi.org/10.1007/s11227-024-06805-3>

Publisher: Springer

Version: Accepted Version

Downloaded from: <https://e-space.mmu.ac.uk/637757/>

Usage rights:  [Creative Commons: Attribution 4.0](https://creativecommons.org/licenses/by/4.0/)

Additional Information: This is an author-produced version of the published paper. Uploaded in accordance with the University's Research Publications Policy.

Data Access Statement: The data used in this research will be made available upon request. The code used for the experiments will be made available upon request.

Enquiries:

If you have questions about this document, contact openresearch@mmu.ac.uk. Please include the URL of the record in e-space. If you believe that your, or a third party's rights have been compromised through this document please see our Take Down policy (available from <https://www.mmu.ac.uk/library/using-the-library/policies-and-guidelines>)

A storage-efficient learned indexing for blockchain systems using a sliding window search enhanced online gradient descent

Emmanuel Acheampong Asiamah^{1,2} · Nana Kwadwo Akraasi-Mensah^{1,2} · Prince Odame^{1,2} · Eliel Keelson^{1,2} · Andrew Selasi Agbemenu^{1,2} · Eric Tutu Tchao^{1,2} · Mohammed Al-Khalidi³ · Griffith Selorm Klogo^{1,2}

Abstract

With its promise of transparency, security, and decentralization, blockchain technology faces significant challenges related to data storage and query efficiency. Current indexing methods, which often rely on structures like Merkle trees and Patricia tries, contribute to excessive storage overhead and slower query responses, particularly for full nodes that maintain a complete copy of the blockchain. To address this, we introduce a novel-learned indexing approach for blockchain that utilizes a layered structure with a sliding window search enhanced Online Gradient Descent (SWS-OGD) as the inter-block index. The method was implemented across five distinct blockchain environments—Bitcoin, Ethereum, Dogecoin, Litecoin, and IoTeX. Experimental results demonstrate that the proposed method reduces storage costs by up to 99% compared to state-of-the-art approaches, requiring as little as 0.9 KB for 20,000 blocks—a substantial improvement over existing methods. Despite the significant reduction in storage costs, the SWS-OGD method maintains comparable performance in other key metrics, such as query latency. These results ensure that blockchain systems can handle large-scale data queries efficiently, maintaining high performance even as the blockchain grows in size.

Keywords Blockchain · Learned index · Online learning · Querying

1 Introduction

Blockchain technology has revolutionized the landscape of digital transactions and data management, providing unprecedented levels of transparency, security, and decentralization [1]. Originating as the underlying technology for Bitcoin, blockchains are now applied in a wide array of sectors beyond cryptocurrencies.

The fundamental structure of a blockchain is a distributed ledger where transactions are securely linked in blocks, ensuring data integrity and immutability through cryptographic principles [2]

Blockchain's importance lies in its ability to provide a reliable and tamper-proof record of transactions without a central authority. This decentralized nature means no single entity controls the entire system [3]. Blockchain technology also secures sensitive data in sensor networks, as demonstrated by solutions leveraging cryptographic accumulators for secure data storage [4], thereby protecting IoT infrastructure from potential vulnerabilities. This has enabled the use of transparent supply chains in which every product's journey can be tracked from origin to consumer [5], and healthcare records are securely shared among authorized parties without the risk of unauthorized access or alteration [6]. Blockchain also facilitates new business models and efficiencies by enabling smart contracts, which are self-executing contracts with the terms directly written into code, automating and enforcing agreements without intermediaries [7]. In developing countries, blockchain could address key issues such as reducing land disputes by providing an immutable record of land ownership, enabling efficient tracking of disease spread and secure sharing of medical records in public health crises, and combating corruption through enhanced transparency and accountability [8–10].

Despite the tremendous potential blockchain has in revolutionizing society positively, one of the challenges to the widespread adoption of blockchain is the significant storage requirements. Since blockchain data cannot be altered or deleted, every transaction is permanently recorded, leading to an ever-increasing volume of data that every node in the network must store. This growing storage demand can quickly become burdensome, particularly for full nodes that must maintain a complete copy of the blockchain. As the size of the blockchain grows, so do the costs and complexities associated with storing and managing this data [11]. This challenge highlights the need for solutions that can optimize storage efficiency without compromising the integrity and security of the blockchain. Addressing these storage challenges is crucial for enabling the broader adoption of blockchain technology, ensuring it remains scalable and sustainable as it continues to evolve and expand its applications across various sectors.

Traditional indexing structures, such as B-trees [12] and Merkle Patricia Tries (MPT) [13], which are often employed to aid efficient querying, exacerbate these storage challenges. While effective for smaller-scale applications, these data structures exhibit linear growth in storage complexity, often approximated by $O(N \times M)$, where N is the number of nodes and M is the size of each node. In rapidly growing blockchain systems with millions of blocks, this linear growth becomes unsustainable. The ever-expanding storage requirements not only increase costs but also hinder efficient query performance, leading to higher latency and straining computational resources. Furthermore, the continuous addition of new data in blockchain environments necessitates frequent index updates, which incur additional computational and memory overhead [14]. Addressing these inherent inefficiencies in traditional indexing methods is critical to unlocking the full potential of blockchain technology and ensuring its scalability across diverse applications.

These scalability issues are particularly pressing given the rate at which blockchain sizes are growing. For instance, by 2014, the Bitcoin blockchain had reached 20 GB, containing around 15,846 blocks with an average block size of 1.3 MB. By July 2024, this blockchain had grown to exceed 580 GB, equating to approximately 417,477 blocks [15]. Projecting forward, it is estimated that by 2034, the Bitcoin blockchain could expand to 1040 GiB (or about 806,597 blocks) and further to 1550 GiB (around 1,219,231 blocks) by 2044. These projections underscore the urgent need for novel approaches to indexing that can efficiently manage the exponential growth of blockchain data.

However, current indexing methods fall short of meeting this challenge. While they aim to enhance query performance, they come with excessive storage demands, rendering them impractical for large-scale systems. Addressing this gap, we propose a novel indexing approach that minimizes storage costs while maintaining comparable query latency to existing state-of-the-art indexing methods.

The following is a summary of the main contributions of this paper:

1. This research introduces a novel learned indexing approach for blockchain indexing that utilizes a layered structure with a Sliding Window Search-enhanced Online Gradient Descent (SWS-OGD). This method focuses on optimizing the inter-block indexing process to efficiently map timestamps to block heights, significantly improving the efficiency of blockchain data retrieval.
2. The proposed learned index method significantly reduces the storage requirements for blockchain systems. By minimizing the index size, the method addresses one of the critical challenges in blockchain technology—high storage overhead. This reduction in storage consumption is crucial for the scalability and efficiency of blockchain applications.
3. Besides the reduced storage requirements, the proposed method maintains comparable performance in key metrics such as query latency. This balance between storage efficiency and query performance ensures that the blockchain system remains both scalable and efficient, providing fast and reliable data retrieval.
4. The proposed method was rigorously evaluated against state-of-the-art learned indexes in a blockchain environment. Experimental results demonstrate that the proposed method significantly outperforms existing methods in terms of storage cost while maintaining or enhancing query efficiency. This validation underscores the practical applicability and effectiveness of the learned index approach in real-world blockchain systems.

The remaining sections are organized as follows: Section II reviews related work on blockchain indexing and querying techniques, highlighting advancements, limitations, and the evolution of methodologies used by other researchers in the field. Section III outlines the detailed methodology for developing and implementing the proposed solution. Section IV presents the results and discussions. Section VI summarizes the key findings of the research and offers recommendations for future work.

2 Related works

Efficient querying in blockchain systems has been approached through various schemes, which can be categorized into four primary schemes: External Database Integration, On-chain Indexing, Smart Contract Querying, and Data Structure Modification.

2.1 External database integration

This scheme involves connecting blockchain systems with external databases to enhance query capabilities by leveraging advanced database functionalities. Middleware approaches use integrations with external databases such as MongoDB and ForkBase, enabling support for complex queries like range and top-k queries, thus significantly improving performance [16–19]. Other strategies apply big data techniques, such as Map/Reduce, to enable efficient data extraction and analysis [20].

Further, Blockchain Database approaches adapt database architecture to blockchain principles. For example, BlockchainDB employs shared tables and data sharding [21], EthernityDB integrates a lightweight database with Ethereum using BSON for efficient storage [22], and HBasechainDB leverages the Hadoop ecosystem to enhance data storage and retrieval [23].

While these methods improve querying capabilities, they often introduce increased storage requirements and system complexity. These approaches generally involve redundant data storage across both the blockchain and external databases; BlockchainDB and HBasechainDB, for instance, require additional distributed data structures and sharding, further inflating storage overhead.

2.2 Smart contract querying

Alternative approaches to enhance data retrieval have been implemented in specific blockchain applications, including ride-sharing, pharmacogenomics, and SQL query processing. These methods integrate smart contracts with data structures like the Merkle Patricia Trie and employ upgradable contracts to support evolving requirements [24–28]. While these solutions improve data accessibility and flexibility, they also impose additional storage demands, as each instance or update of a smart contract is permanently recorded on the blockchain. The use of complex structures, such as the Merkle Patricia Trie, combined with the need to store multiple contract versions for future upgrades, further increases both storage consumption and computational complexity, creating scalability challenges for blockchain systems.

2.3 Data structure modification

Data Structure Modification involves altering the fundamental data structures of the blockchain to facilitate faster information retrieval. Proposed modifications include the Multi-State Merkle Patricia Trie (MSMPT), which enhances key-based searches with linked-list storage [29], the integration of height-balanced Binary Search Trees

with Threaded Binary Search Trees for faster searches [30], and restructuring blocks into index and data layers using Abstract-Trie and Operation-Record List [31]. While these modifications enhance query efficiency, they typically result in higher storage demands due to additional indexing layers and metadata structures.

2.4 On-chain indexing

On-chain indexing refers to creating and maintaining indexing structures within the blockchain itself to enhance the efficiency of data retrieval processes. Unlike external database solutions, on-chain indexes store data structures directly on the blockchain, thus preserving the decentralized nature of blockchain systems. These indexing structures aim to improve the speed of point and range queries-critical for applications like transaction validation, smart contract execution, and historical data retrieval.

These indexing structures significantly reduce search complexity in retrieving data from large and growing blockchain ledgers. For example, an index can eliminate the need to sequentially traverse the entire blockchain to locate a specific transaction or set of blocks, thus improving the efficiency and speed of queries.

Various techniques have been proposed in this domain, each tailored to different aspects of blockchain querying. Hybrid index systems, for example, combine B-trees and Skip-lists to improve access times by balancing tree depth and node traversal paths [32]. Some methods embed index data within individual transactions to enable faster data lookup, essentially coupling data and index at the transaction level [33].

Advanced structures like the Group Merkle Patricia Tree (GMPT) [34] cluster blockchain accounts using Merkle Patricia Trees combined with K-Means clustering to optimize query efficiency by reducing verification time. Similarly, the Adaptive Balanced Merkle (AB-M) Tree [13] enhances storage scalability and query speed by combining rapid retrieval and data verification mechanisms. The Authenticated Layered Index (ALI) [35] employs a hybrid on-chain and off-chain model to support efficient queries for lightweight clients. The EBTree structure [12] uses a hierarchical metadata-based design to facilitate efficient traversal and querying in Ethereum. The Deterministic Append-only Skip List (DASL) [36] integrates Merkle DAG structures to improve provenance query efficiency while maintaining minimal runtime overhead.

The Merkle Semantic Trie (MST) [37] introduces real-time querying capabilities, supporting complex query types such as semantic and range queries without altering the underlying blockchain database. The Subchain-based Account Transaction Chain (SCATC) [38] divides account transaction histories into smaller, hash-linked subchains to improve query performance for accounts with extensive histories. The vChain [39] utilizes a Sliding Window Accumulator (SWA) within an authenticated data structure to enhance dynamic query efficiency, particularly for large-scale implementations. Finally, BCTkPQ [40] employs a Blockchain Transaction Graph (BTG) with collaborative query parsing and execution to efficiently handle first-‘k’ query paths with high accuracy.

While traditional on-chain indexing techniques can enhance query performance, they often impose substantial storage costs due to the need to maintain complex data structures and metadata on the blockchain. This storage overhead scales with the number of transactions or blocks, ultimately impacting blockchain scalability and sustainability. These limitations underline the necessity for alternative approaches that achieve high query efficiency without the excessive storage demands characteristic of on-chain indexing.

2.5 Learned indexes

Learned indexes represent a promising alternative by employing machine learning models to predict data locations within the blockchain. Unlike traditional on-chain indexing, which stores metadata and extensive structures directly on-chain, learned indexes leverage predictive modeling, allowing for efficient data access with minimal storage requirements.

The concept of learned indexes, introduced by Kraska et al. [41], proposes replacing traditional index structures with machine learning models that can learn and exploit data distributions. By doing so, learned indexes can significantly improve query performance and storage efficiency. Learned indexes can significantly reduce the memory footprint compared to traditional indexes. For example, by modeling the data's cumulative distribution function (CDF), a learned index can store the mapping information more compactly.

Since their inception, learned indexes have sparked significant research interest, leading to various improvements and adaptations. A notable advancement in this domain is presented by Ding et al. [42]. ALEX addresses the limitations of static learned indexes by introducing an adaptive layout that efficiently handles dynamic updates such as inserts, deletes, and updates. This approach ensures the index remains efficient even as the data distribution changes over time. Experimental results show that ALEX achieves up to 4.1× higher throughput and up to 2000× smaller index size than B+Trees. It also outperforms the original learned index in read-only scenarios while maintaining a smaller index size.

Ge et al. [43] presented SALI, which incorporates adaptive strategies to handle various workload skews, enhancing concurrency performance. SALI incorporates adaptive strategies to handle multiple workload skews, enhancing concurrency performance. The experimental results showcased that SALI improved insertion throughput by an average of 2.04× with 64 threads compared to the second-best learned index, ALEX+.

2.6 Learned indexes in blockchain

In the context of blockchain, learned indexes have been explored to address the unique challenges of blockchain's immutable and append-only nature. Zhang et al. [44] introduced COLE, a column-based learned storage for blockchain systems, which leverages learned models to index historical state values efficiently. COLE addresses the high storage costs and ensures data integrity through

a combination of column-based design and learned indexes optimized for disk environments. While COLE offers substantial storage size reductions and query performance improvements compared to traditional indexing methods like Merkle Patricia Trie (MPT), its focus is primarily on provenance queries, which limits its applicability to more common and critical query types like point and range queries.

Another significant work is by Yao et al [45], who proposed a learned-index-based semantic keyword query architecture for blockchain. This architecture records data semantics information to support efficient semantic keyword queries, establishing a lookup table index for semantic information among blocks and a block-level recursive model index to improve query efficiency. By storing the lookup table in extended block headers and maintaining recursive model indexes off-chain, the proposed system enhances query performance while ensuring the completeness and correctness of query results. The experimental results show that combining the lookup table and the learned index effectively improves query efficiency on the blockchain, demonstrating substantial improvements in query speed and storage efficiency. However, this approach is specifically tailored for semantic queries and does not address the challenges associated with optimizing point and range queries, which are vital for most blockchain applications.

In the recent study presented by Chang et al. [46], the authors propose a novel approach to indexing in blockchain systems named Anole. This approach leverages learned indexes to optimize point and time-range queries, significantly improving performance and storage efficiency compared to traditional methods.

Anole employs a dynamic piecewise linear regression approach, which fits well within the online learning framework. Online learning is a framework for designing and analyzing algorithms that build predictive models by processing data sequentially. This approach is particularly efficient for large datasets, as it updates models incrementally with each new data point rather than retraining from scratch with the entire dataset [47].

While Anole reduces storage overhead compared to traditional methods, its use of dynamic piecewise regression in the learned inter-block index still requires considerable storage to maintain the parameters of the linear functions. Our approach distinguishes itself by employing SWS-OGD, which does not require segment-based model storage, thereby reducing the storage cost significantly while maintaining comparable performance in query latency and accuracy (Figure 1).

3 System overview

This section provides an overview of the system architecture for the proposed learned index for blockchain systems. The system consists of full nodes, each maintaining a layered learned index to efficiently manage and retrieve blockchain data. Our proposed solution is designed to be compatible with various blockchain consensus algorithms, including innovative mechanisms like Reputation Awareness Randomization Consensus [48].

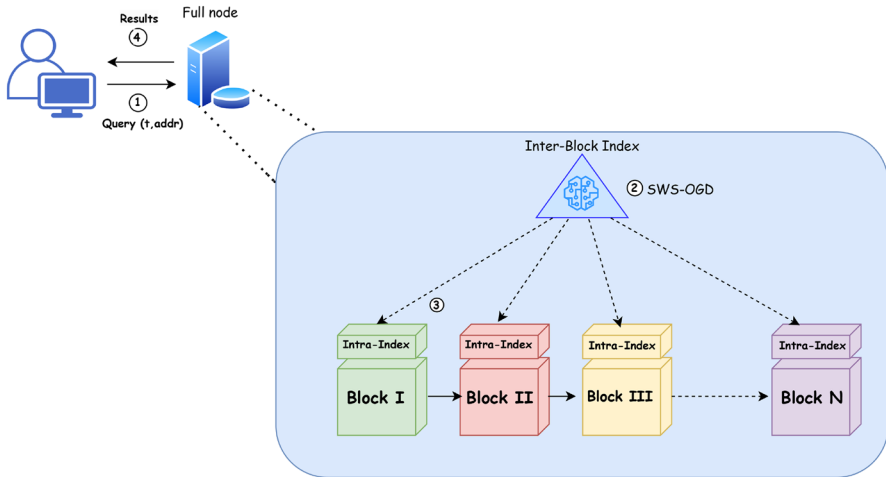


Fig. 1 System overview of the proposed learned index architecture

3.1 System components

The system architecture comprises these key components, as can be seen in Fig. 1:

1. **Full nodes:** Full nodes are crucial components of the blockchain network, maintaining a complete copy of the blockchain ledger. They are responsible for validating and relaying transactions to ensure the integrity and security of the blockchain. Full nodes store block headers and transaction data, processing query requests and managing data updates. Full nodes in the system utilize the proposed layered learned index without additional indexing schemes. All queries are executed on the full nodes, leveraging the blockchain's inherent security measures for user connections, which are assumed to be in place.
2. **Layered learned index:** This index structure is designed to optimize point and range queries using a hierarchical model. The top layer handles inter-block indexing, which identifies the relevant blocks for a query. The lower layer manages intra-block indexing within those identified blocks, ensuring that both indexes work together seamlessly to provide efficient and precise data retrieval.
 - **Inter-block indexing:** Focuses on relationships and data distribution across blocks within the blockchain. This type of indexing is used to quickly locate which blocks contain the data relevant to a query, significantly narrowing down the search space before performing a more detailed search within the blocks.
 - **Intra-block indexing:** Focuses on the data within individual blocks. Once the inter-block index identifies the relevant blocks, the intra-block index helps locate the exact data points within those blocks.

3. **Clients (Users):** Clients, also called users, interact with the blockchain system by submitting queries to full nodes. They are integral to the system as they initiate the data retrieval process.

3.2 Proposed inter-block indexing algorithm

3.2.1 Problem formulation and objective

The inter-block index is designed to efficiently map timestamps to block heights, enabling quick and accurate queries in a blockchain system. This section formulates the problem and sets the objective for the inter-block indexing algorithm.

Problem formulation: Consider a sequence of data points $\{(x_i, y_i)\}_{i=1}^N$, where:

- x_i : the normalized timestamp representing the time at which a transaction occurred, scaled to a uniform range.
- y_i : the corresponding block height, representing the position of the block in the blockchain.

The challenge is to build an index that accurately predicts the block height y_i for a given timestamp x_i with high efficiency.

Objective: The objective is to minimize the prediction error across all data points, aiming to accurately predict y_i given a timestamp x_i . A linear prediction function $f(w; x_i) = x_i^T w$ is defined, where:

- w : the parameter vector that defines the prediction model.

The goal is to find the optimal parameter vector w that minimizes the cumulative prediction error over all data points. This is done by minimizing the sum of squared errors between the actual and predicted block heights. Let $L(w, x_i, y_i)$ be the loss function, defined as the squared error between y_i and $x_i^T w$:

$$\min_w \sum_{i=1}^N L(w, x_i, y_i) = \min_w \sum_{i=1}^N (y_i - x_i^T w)^2 \quad (1)$$

This objective ensures that the predicted block heights are as close as possible to the actual block heights, improving the accuracy of the inter-block index in mapping timestamps to block heights. The Online Gradient Descent (OGD) algorithm, enhanced with a sliding window search, is proposed to solve this optimization problem.

3.2.2 Online gradient descent (OGD)

Online Gradient Descent (OGD) is specifically tailored to address sequential learning problems, where data points arrive one at a time, and the model must be updated in real-time without reprocessing the entire dataset [49]. This characteristic makes it

particularly suitable for blockchain systems, which generate new blocks sequentially over time.

Unlike traditional batch gradient descent, which processes the entire dataset at once to compute the gradient and update the model parameters, OGD updates the model incrementally with each new data point as shown in Fig. 2.

To minimize the objective function, Online Gradient Descent computes the gradient of the loss function with respect to the model parameters w . This gradient indicates the adjustment needed to reduce the difference between the predicted and actual block heights. The gradient of the loss function $L(w, x_i, y_i)$ with respect to w is:

$$\nabla_w L(w, x_i, y_i) = -2x_i(y_i - x_i^T w) \tag{2}$$

where:

- $\nabla_w L(w, x_i, y_i)$: the gradient of the loss function with respect to the parameter vector w .
- $-2x_i$: the partial derivative of the squared error term with respect to w .
- $y_i - x_i^T w$: the error term representing the difference between the actual block height y_i and the predicted block height $x_i^T w$.

Update rule: The update rule in OGD adjusts the model parameters iteratively to reduce the loss function value:

$$w_{t+1} = w_t - \eta \nabla_w L(w_t, x_i, y_i) = w_t + 2\eta x_i(y_i - x_i^T w_t) \tag{3}$$

where:

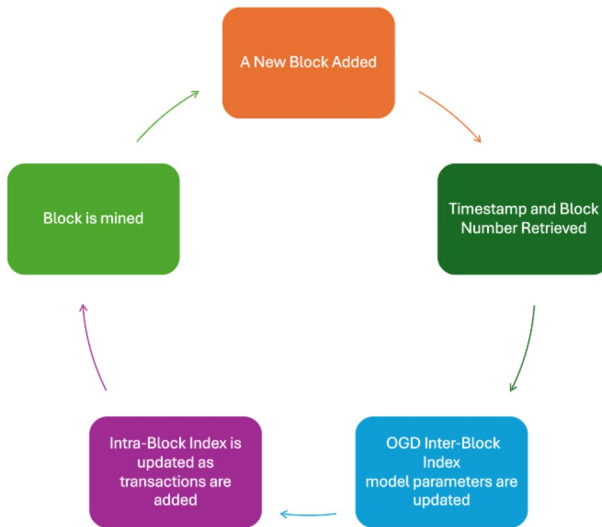


Fig. 2 Iterative process of model and index updates

- w_{t+1} : the updated parameter vector after iteration t .
- w_t : the parameter vector at iteration t .
- η : the learning rate, which determines the step size of each update.
- $2\eta x_i(y_i - x_i^T w_t)$: the adjustment to w based on the gradient.

By iteratively applying this update, the parameters converge to values that minimize the prediction error. This process is tied directly to the objective function, aiming to minimize cumulative prediction error:

$$\min_w \sum_{i=1}^N L(w, x_i, y_i) = \min_w \sum_{i=1}^N (y_i - x_i^T w)^2 \quad (4)$$

Using Online Gradient Descent to iteratively update w minimizes this cumulative loss, achieving the goal of accurately predicting block heights from timestamps.

3.2.3 Assumptions

In developing the proposed learned index algorithm for blockchain systems, several assumptions are made to ensure convergence and performance:

- **Convex loss function:** The MSE loss function is assumed to be convex with respect to w , which guarantees that any local minimum is a global minimum. This is crucial for the optimization process, as it allows the algorithm to converge reliably to the optimal solution.
- **Lipschitz continuity:** It is assumed that the gradients are Lipschitz continuous, meaning there exists a constant L such that:

$$\|\nabla L(w) - \nabla L(w')\| \leq L\|w - w'\| \quad (5)$$

This assumption ensures that the gradient does not change abruptly, stabilizing the optimization process and providing consistent updates to the model parameters.

- **Bounded gradients:** The gradients are assumed to be bounded by a constant G :

$$\|\nabla L(w)\| \leq G \quad (6)$$

To prevent excessively large gradient values, gradient clipping is applied by setting a maximum threshold G for the gradient norm. This ensures stability in the optimization process by scaling down gradients that exceed this threshold, preventing destabilization and promoting steady convergence.

- **Diminishing step size:** The step size η_t diminishes over time, satisfying:

$$\sum_{t=1}^{\infty} \eta_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \eta_t^2 < \infty \quad (7)$$

This ensures that updates become smaller as the algorithm progresses, allowing it to converge to the minimum loss.

These assumptions support the convergence and stability of the proposed learned index algorithm, ensuring accurate mapping of timestamps to block heights with minimal cumulative prediction error.

3.2.4 Sliding window prediction mechanism

The sliding window prediction mechanism refines initial predictions by incrementally adjusting the search range until the correct value is found. Unlike fixed-range searches, this sliding window dynamically expands its range, effectively "sliding" to cover a larger area if the correct value is not located within the initial range. This adaptive approach ensures completeness and is particularly suited to block generation times with high variability.

The sliding window process begins with an initial prediction, generated by the Online Gradient Descent (OGD) model. This prediction is calculated based on the normalized input data and then scaled back to the original space of the target variable, such as block heights. The initial prediction serves as a starting point for the search.

Based on the initial prediction, an initial search range, denoted as w_s , is centered around the predicted index, which corresponds to the closest match in the block heights array. This range serves as the immediate neighborhood within which the search will begin.

The search proceeds bidirectionally within this initial range. Starting from the predicted index, the search iterates forward and backward in steps, defined by a parameter s (set to 5 in the implementation). At each step, the algorithm checks whether the timestamp at the current index matches the target timestamp. If a match is found, the correct value is returned, terminating the search.

In cases where no match is found within the initial range w_s , the window expands or "slides" incrementally beyond this predefined range. The algorithm continuously extends the boundaries of w_s until the correct value is found or until the bounds of the block heights are reached. This adaptive sliding mechanism is essential for ensuring completeness, as it guarantees that the query is located, even if it lies outside the initially predicted range.

3.3 Intra-block learned index adoption

The approach leverages Anole's [46] intra-block learned index methodology to enhance querying efficiency within individual blocks. Given that most blockchains, such as Bitcoin, average over 1,000 transactions per block, optimizing query times within these blocks is essential. The purpose of adopting Anole's intra-block learned index is to streamline intra-block query performance.

Construction of intra-block learned index: The intra-block learned index is constructed by organizing transactions within a block according to addresses. For each unique address, the first occurrence of a transaction serves as the aggregation point, grouping subsequent transactions under that address. This ensures that queries

targeting a specific address can quickly access the aggregation point without scanning the entire block.

For example, consider transactions tx_1 and tx_2 associated with address $addr_1$, and transactions tx_3 and tx_4 associated with address $addr_2$. According to this method, tx_1 would be the aggregation point for $addr_1$, while tx_3 would be the aggregation point for $addr_2$. This organization reduces query time within a block by structuring the data to minimize search complexity. Since the number of transactions within a single block is typically limited, this approach ensures that query processing remains efficient and precise.

The construction process involves:

1. Sorting transactions within each block based on the address.
2. Aggregating transactions for each address, with the first transaction serving as the aggregation point.
3. Constructing the intra-block index using the sorted transactions and different aggregation points.

Given the relatively small amount of data within a single block, the intra-block learned index does not require frequent updates. This stability allows the error bound of the intra-block learned index to be set to zero, achieving precise positioning and efficient data retrieval.

3.4 Query process overview

When a client submits a query, the system uses the inter-block index to determine the relevant block heights. It then uses the intra-block index to locate the desired transactions inside the block.

The querying process in the proposed system involves several steps to ensure efficient and accurate retrieval of data from the blockchain. The process is as follows:

1. **Initial query submission:** The client submits a query Q specifying the address and the time range or a specific point in time.
2. **Inter-block index lookup using OGD:** For point queries, the system uses the inter-block index with Online Gradient Descent (OGD) to predict the block height that might contain the transaction. The system predicts the block heights for the start and end times for range queries. A sliding window search mechanism refines these predictions to ensure accuracy.
3. **Intra-block index lookup:** The system uses the intra-block index for each identified block to locate the exact transactions within the block. For a given address, locate the aggregation point in the sorted list of transactions within the block. Once the aggregation point is identified, all transactions associated with that address will be retrieved directly.
4. **Result compilation:** The system aggregates the results before returning them to the client.

Algorithm 1 Point and Range Query with a Learned Index

Require: $Q = (addr, t)$ or $Q = (addr, \langle t_1, t_2 \rangle)$ \triangleright Input query, either a point or range query

Ensure: $TxSet$ \triangleright Output set of transactions matching the query

- 1: Initialize w for OGD \triangleright Initialize weights for Online Gradient Descent (OGD)
- 2: **if** Q is a point query **then** \triangleright Handle single timestamp (point) query
- 3: $(pred_h) \leftarrow \text{OGD}(t)$ \triangleright Predict block height for given timestamp using OGD
- 4: $(pred_h) \leftarrow \text{SlidingWindow}(pred_h, error)$ \triangleright Refine prediction using sliding window to reduce errors
- 5: **for** h in $(pred_h - error)$ to $(pred_h + error)$ **do** \triangleright Search within error range around predicted height
- 6: $Tx \leftarrow \text{intra_fn}(addr, h)$ \triangleright Check for transaction at predicted height
- 7: **if** Tx exists **then** \triangleright If a transaction for the address is found
- 8: $TxSet \leftarrow TxSet.append(Tx)$ \triangleright Add transaction to result set
- 9: **end if**
- 10: **end for**
- 11: **else** \triangleright Handle range query spanning two timestamps
- 12: $(pred_h_1) \leftarrow \text{OGD}(t_1)$ \triangleright Predict block height for start of range
- 13: $(pred_h_2) \leftarrow \text{OGD}(t_2)$ \triangleright Predict block height for end of range
- 14: $(pred_h_1) \leftarrow \text{SlidingWindow}(pred_h_1, error)$ \triangleright Refine prediction for start of range
- 15: $(pred_h_2) \leftarrow \text{SlidingWindow}(pred_h_2, error)$ \triangleright Refine prediction for end of range
- 16: **for** h in $(pred_h_1 - error)$ to $(pred_h_2 + error)$ **do** \triangleright Search across predicted range with error tolerance
- 17: $Tx \leftarrow \text{intra_fn}(addr, h)$ \triangleright Check for transactions within range
- 18: **if** Tx exists **then** \triangleright If a transaction for the address is found within the range
- 19: $TxSet \leftarrow TxSet.append(Tx)$ \triangleright Add transaction to result set
- 20: **end if**
- 21: **end for**
- 22: **end if**
- 23: **return** $TxSet$ \triangleright Return set of transactions that match the query

Algorithm 1 handles both point and range queries. The algorithm predicts a single block height for point queries and searches around this prediction to locate the transaction. For range queries, the algorithm predicts the block heights for the start and end of the time range by effectively performing two-point queries and then searching within the range to locate the transactions. The sliding window search ensures that any potential deviations in the predictions are corrected by examining a localized window around the initial predictions.

3.5 Implementation

The system was implemented on an Intel(R) Core(TM) i5-10500T CPU @ 2.30GHz, with an 8.00 GB (7.78 GB usable) system. A custom blockchain was implemented

using Rust, which was chosen for its performance and safety features. The implementation involved constructing both inter-block and intra-block indexes to facilitate efficient querying. Bitcoin and Ethereum data was collected from Google BigQuery. The block height, input and output addresses, input and output values, and block timestamps for transactions were retrieved. The retrieved data was used to build the blockchain and construct the indexes. In total, 20,000 blocks were created, each containing transactions as they appeared in the Bitcoin, Ethereum, Iotex, Dogecoin, and Litecoin blockchain. These datasets provided a comprehensive view of transaction flows and patterns over a period, facilitating an accurate representation of the blockchain for experimental purposes. Using this data, the efficiency and effectiveness of the proposed indexing algorithm could be tested in real-world scenarios, ensuring that the results are both practical and relevant to actual blockchain environments and also giving two different workloads to test how the proposed method performed.

3.6 Evaluation

A set of evaluation metrics was employed to assess the effectiveness of the proposed inter-block indexing algorithm. These metrics were chosen to provide a holistic view of the algorithm's performance in various aspects crucial for blockchain indexing. The metrics include:

- **Size of the inter-block index:** This metric evaluates the memory efficiency of the index by measuring its total storage consumption. A smaller index indicates better scalability and efficiency, which is essential for blockchain systems with limited storage resources.
- **Average CPU time per update:** This metric measures the average time the CPU takes to update the inter-block index whenever a new block is added to the blockchain. It reflects the computational load required for each update, averaged across multiple instances, and indicates how efficiently the algorithm can handle new data over time.
- **Query latency:** This metric assesses the responsiveness of the index by measuring the time taken to retrieve data in response to query requests. Lower query latency signifies a faster and more efficient data retrieval process, enhancing the overall usability of the blockchain system.

The performance of the proposed algorithm was benchmarked against other online learning algorithms. These include:

- **Anole (Inter-index):** The Anole Inter-Index uses a Dynamic Piecewise Linear Regression, outperforming state-of-the-art on-chain indexing techniques like vChain+. Notably, Anole is the only algorithm used that has been published in the literature.
- **Recursive least squares (Inter-index):** A well-established algorithm known for its precision in parameter estimation. RLS operates by recursively updating its model parameters with each new data point, making it highly suitable for online

learning applications. It adapts to changes in real-time while efficiently managing memory, thus providing accurate and stable predictions in dynamic environments. RLS was chosen for benchmarking due to its strong reputation for reliability in adaptive filtering and online learning contexts.

A comparison against traditional indexing methods was not made because Anole had already demonstrated superiority in all metrics. Additionally, a comparison against any other learned indexes in blockchain apart from Anole was not made because the two other learned indexes were used for entirely different purposes. Anole is the only learned index used for point and range queries within the blockchain context.

4 Results and discussion

4.1 Storage cost results

The storage cost is the most important metric as it relates directly to the aim of this paper. The model size in kilobytes (KB) measures the storage cost. Figure 3 and Fig. 4 show that the Anole Inter-Index algorithm has a significantly higher storage cost than SWS-OGD and RLS. As the number of blocks increases, the storage cost of Anole Inter-Index grows linearly, reaching approximately 100 KB at 20,000 blocks. In contrast, the storage costs of SWS-OGD and RLS remain almost constant, at around 0.12 KB and 0.28 KB, respectively. This indicates that SWS-OGD and RLS are more efficient in terms of storage.

The minimal storage cost of SWS-OGD arises from its design philosophy, which centers on incremental parameter updates rather than creating new models. In SWS-OGD, a single set of model parameters is continuously refined as new blocks are added. This approach is inherently storage-efficient: the model's

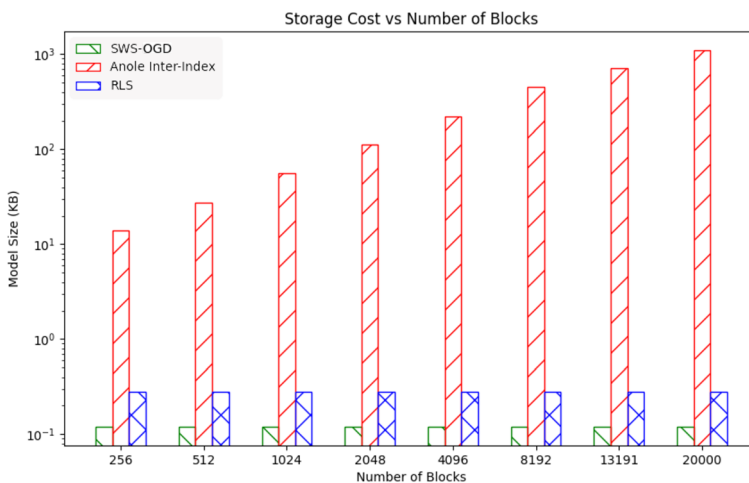


Fig. 3 Index size in bitcoin environment

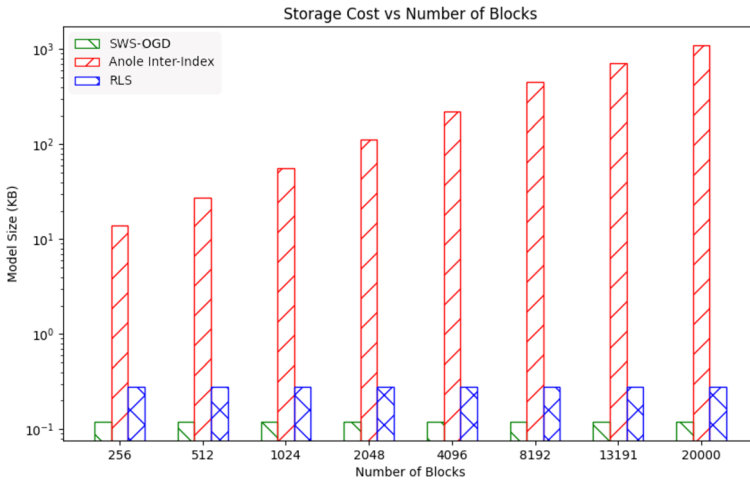


Fig. 4 Index size in Ethereum environment

size remains stable because it does not need to store parameters for different data segments, in contrast to Anole Inter-Index. Anole Inter-Index employs a dynamic piecewise regression, which creates multiple local models for different data segments to optimize prediction accuracy. While effective in this regard, it significantly increases storage cost linearly as each additional segment requires new model parameters and will require lots of space as the blockchain grows. Although RLS also aims for storage efficiency, it maintains a covariance matrix, which introduces additional storage overhead. This matrix is essential for its recursive update mechanism, which estimates parameters based on historical data and continuously updates them as new data is introduced.

Notably, the storage cost of Anole Inter-Index is directly influenced by the variability of the data it models. For datasets with high variability in block generation times, such as those found in certain blockchain systems, Anole Inter-Index would require a larger number of segments to maintain accuracy, thereby increasing its storage footprint substantially.

SWS-OGD is uniquely suited for environments with variable or high-frequency data, such as blockchain systems, due to its storage independence from data variability. Unlike Anole, where storage costs scale with both data size and variability, SWS-OGD remains unaffected by these factors, maintaining a consistent storage footprint that is ideal for large-scale, high-throughput systems.

We report storage cost results for only Bitcoin and Ethereum workloads as representative datasets because the storage performance of indexing algorithms is not significantly affected by the underlying dataset. This is due to the design of SWS-OGD, which updates model parameters incrementally and does not rely on dataset-specific characteristics such as block interval variability. Thus, the trends observed with Bitcoin and Ethereum workloads are consistent across other datasets. Reporting these two allows for a concise presentation without redundancy while maintaining the generalizability of the findings.

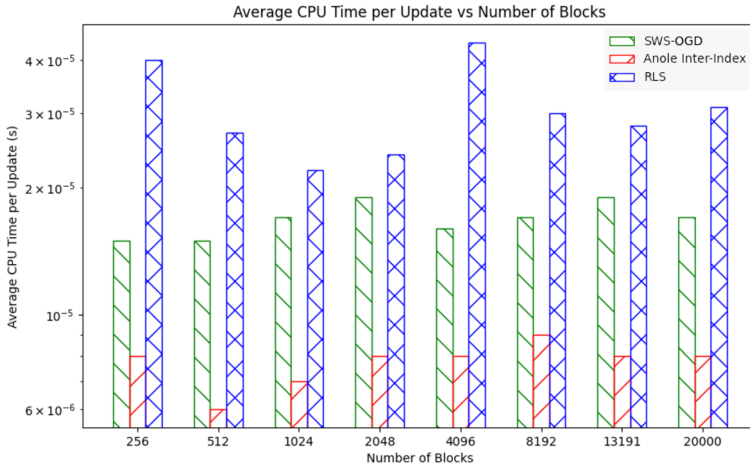


Fig. 5 Average CPU time for bitcoin workload

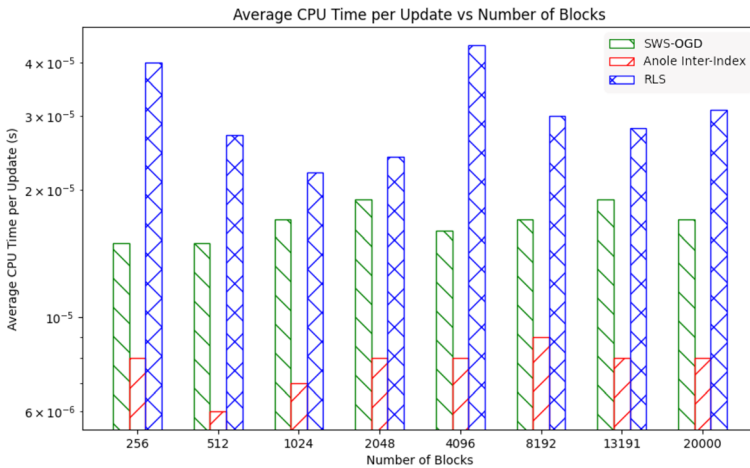


Fig. 6 Average CPU time for Ethereum workload

4.2 Average CPU time per update results

The average CPU time per update measures the computational efficiency of the algorithms. It is crucial for maintaining real-time performance in dynamic blockchain environments. The results shown in Fig. 5 and Fig. 6 indicate that SWS-OGD has a lower average CPU time per update than RLS but slightly higher than Anole Inter-Index. Specifically, SWS-OGD maintains an average CPU time per update of around 2×10^{-5} seconds, while RLS reaches up to 8×10^{-5} seconds at 4,096 blocks. Anole Inter-Index remains the most efficient in terms of CPU time, with values consistently below 1×10^{-5} seconds.

In terms of computational efficiency, SWS-OGD shows a balanced performance in CPU time per update. Although Anole Inter-Index exhibits lower CPU times due to its approach of creating new model parameters without constant training, this comes at the cost of higher storage usage and increased complexity. SWS-OGD, on the other hand, takes more CPU time per update because it refines predictions and updates the same model parameters across the entire dataset, which involves fitting the model continuously. This continuous fitting process, while more computationally intensive, ensures that the model remains accurate and smaller as the blockchain grows. For future work, exploring methods to optimize the computational efficiency of SWS-OGD, such as parallelizing updates or introducing more efficient fitting techniques could further enhance its applicability in large-scale blockchain systems.

For computational efficiency, results are reported for two representative datasets- Bitcoin and Ethereum- because the outcomes remain consistent across datasets. The efficiency of SWS-OGD is primarily influenced by its design and not by dataset-specific characteristics, as the computational load depends on the number of blocks and updates rather than the inherent variability of the dataset. This consistency across datasets validates the generalizability of the method.

For future work, exploring methods to optimize the computational efficiency of SWS-OGD, such as parallelizing updates or introducing more efficient fitting techniques could further enhance its applicability in large-scale blockchain systems.

4.3 Latency analysis

The latency comparison in Figs. 7, 8, 9, 10, and 11 highlights the performance of SWS-OGD, Anole, and RLS across various checkpoints. The proposed SWS-OGD method, represented by green bars, demonstrates a latency performance comparable

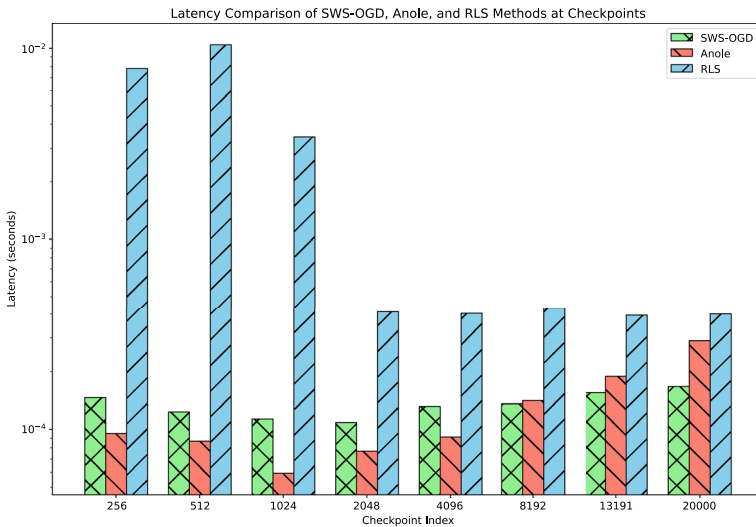


Fig. 7 Bitcoin latency comparison

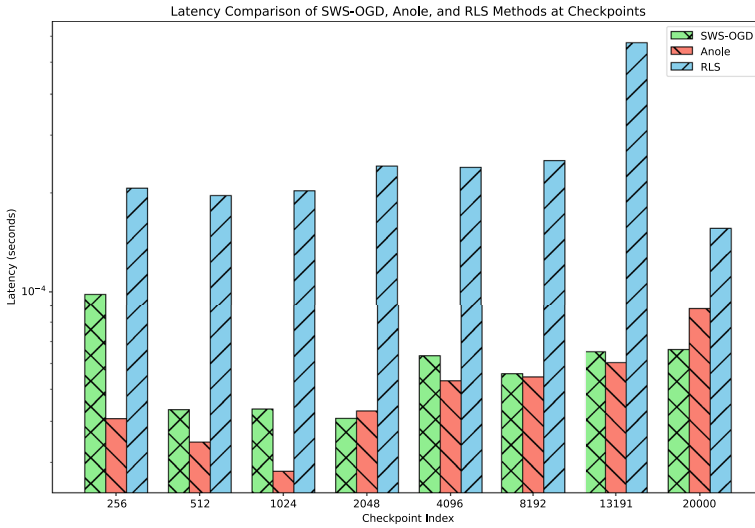


Fig. 8 Ethereum latency comparison

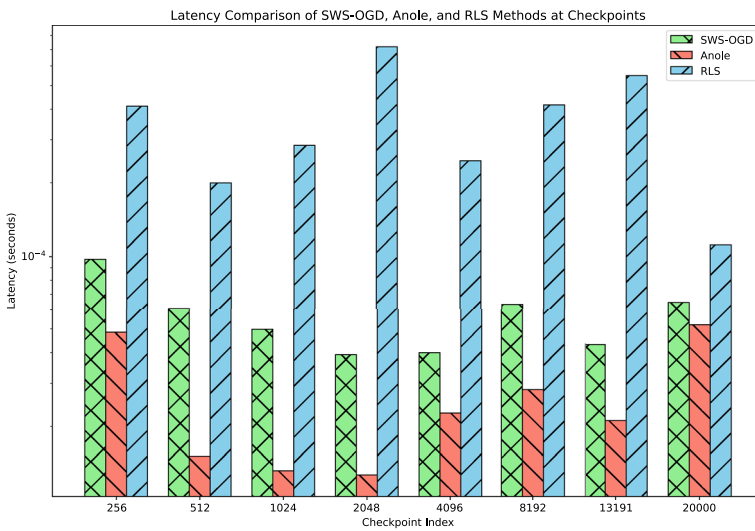


Fig. 9 IoTeX latency comparison

to Anole (red bars) while significantly reducing storage costs, as previously discussed. This observation validates SWS-OGD’s efficiency in achieving low latency with minimal storage, positioning it as a viable alternative to Anole, particularly in storage-constrained environments.

At early checkpoints (e.g., 256 and 512), SWS-OGD shows slightly higher latency than Anole. This higher initial latency is attributed to the limited number of

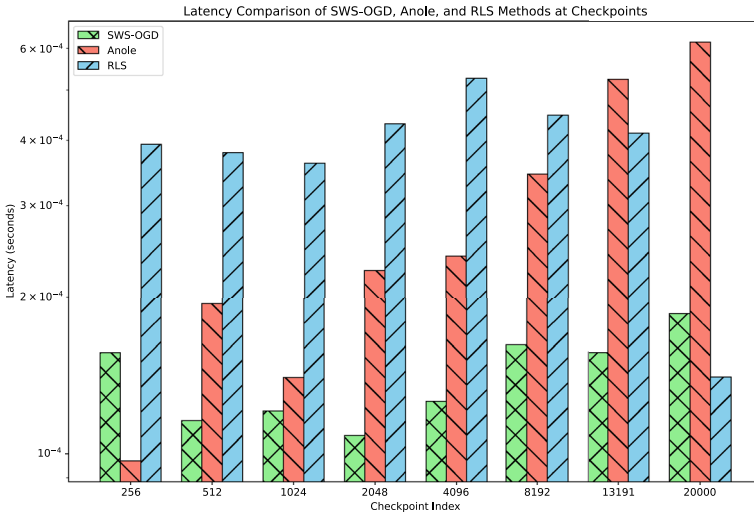


Fig. 10 Litecoin latency comparison

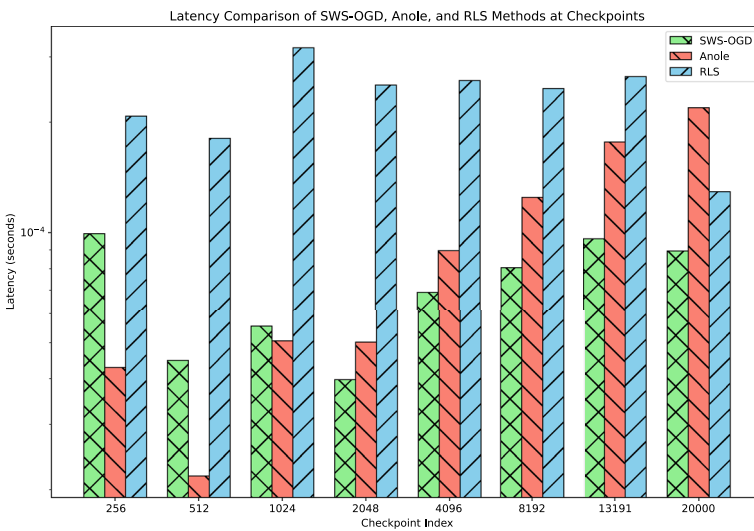


Fig. 11 Dogecoin latency comparison

entries in the model at these early stages, leading to an incomplete fit. As more data becomes available and the model refines its predictions, SWS-OGD’s latency stabilizes and becomes comparable to Anole, demonstrating its ability to adapt efficiently as the dataset grows.

Anole exhibits an incremental rise in latency as the checkpoint index advances. This increase can be attributed to Anole’s dynamic segmentation approach, where additional segments may be created over time to maintain prediction accuracy,

inadvertently adding latency due to segment management overhead. This trend indicates that Anole’s latency performance is sensitive to the complexity of the dataset, especially as variability in block generation times increases with higher block indices.

RLS consistently records the highest latency across all checkpoints. Its latency remains elevated and even increases at later checkpoints. This can be attributed to the recursive updates and covariance matrix adjustments, which add computational overhead, making RLS the least suitable for real-time applications in blockchain indexing.

Notably, SWS-OGD and Anole converge in latency at higher checkpoints (e.g., 8192 and 13191), where both methods exhibit similarly low latency values. This convergence suggests that SWS-OGD’s adaptive sliding window efficiently minimizes latency as the model becomes more stable over time, aligning closely with Anole’s performance but with the advantage of reduced storage requirements.

4.3.1 Impact of block interval variability on latency performance

The datasets used in this analysis—Bitcoin, Ethereum, Dogecoin, Litecoin, and IoTeX—exhibit significant differences in block interval characteristics, as summarized in Table 1. These interval statistics provide essential context for interpreting the latency trends observed in the Latency Comparison. Variability in block generation times influences how the SWS-OGD, Anole, and RLS algorithms adapt and respond, impacting their latency performance at different checkpoints.

The datasets exhibit a broad range of mean intervals and variabilities, with Ethereum and IoTeX showing particularly high standard deviations (46916.22 s and 28714.89 s, respectively), which indicate extreme fluctuations in block generation times. Conversely, Bitcoin and Dogecoin show relatively low variability, with standard deviations of 588.21 s and 76.68 s, respectively. This diversity in block interval variability is crucial, as it presents different challenges for indexing algorithms that rely on temporal consistency.

Despite these differences, SWS-OGD demonstrates effective latency performance across all datasets. The algorithm’s ability to adapt its sliding window search dynamically allows it to maintain low latency even when the underlying data distribution exhibits high variability, as seen with Ethereum and IoTeX. By adjusting the search

Table 1 Block interval statistics for different datasets

Dataset	Mean interval (s)	Std dev (s)	Min interval (s)	Max interval (s)
Ethereum	388.82	46916.22	1	6566428
Bitcoin	588.76	588.21	0	6902
Dogecoin	69.12	76.68	0	712
Litecoin	263.42	603.12	0	15166
IoTeX	211.69	28714.89	5	4060805

range based on the initial prediction error, SWS-OGD compensates for unexpected fluctuations in block intervals, ensuring that query response times remain consistent.

4.4 Success rate of sliding window search across datasets

The success rate of the sliding window search mechanism was evaluated across multiple blockchain datasets, including Bitcoin, Ethereum, Dogecoin, Litecoin, and IoTeX. Success rates were calculated for increasing search ranges, from 1 up to when the rates hit hundred percent or close, and the results were plotted to observe how the search range impacts accuracy.

The sliding window search range w_s significantly affects the ability of the model to locate the correct target within the specified search radius. As shown in Fig. 12, there is a clear trend across datasets: as the search range w_s increases, the success rate improves. This is expected, as a larger w_s allows the model to consider a broader range of indices, thereby increasing the likelihood of finding the correct target even if the initial prediction deviates. However, this improvement comes at the cost of higher computational latency, as a larger window size requires additional comparisons.

Although all datasets show an increase in success rate with larger w_s , the rate of improvement varies. For instance, Ethereum and IoTeX datasets achieve near-complete success (100%) at relatively smaller search ranges compared to others. This suggests that the underlying data distribution and volatility within each dataset impact how effectively the sliding window can correct initial predictions. Data with higher volatility or irregular patterns may require a larger search range to achieve a similar success rate.

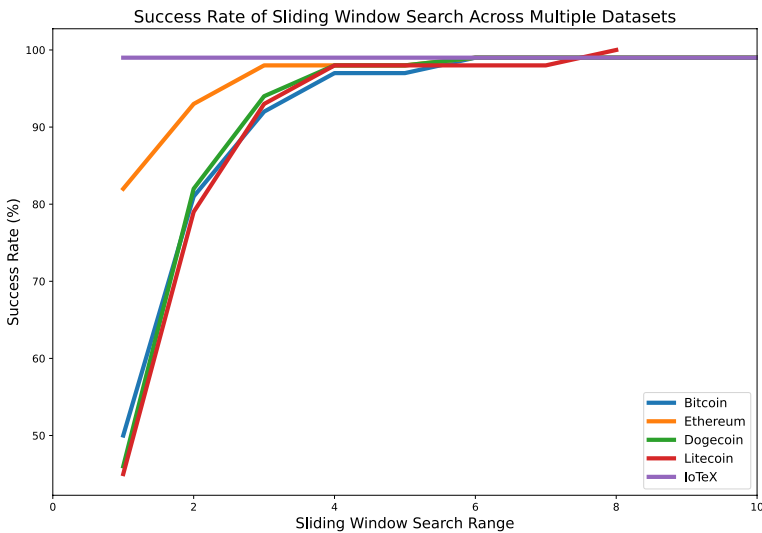


Fig. 12 Success rate of sliding window search across multiple datasets for varying search ranges

Selecting an optimal window size is crucial for balancing accuracy and latency. Smaller search ranges provide lower latency but can result in suboptimal accuracy, as seen with ranges below 5. Conversely, while larger ranges (approaching 10 or more) ensure a high success rate, they incur a latency penalty. Thus, an adaptive approach to setting w_s , potentially based on error measurements during training, may offer an efficient compromise for real-time applications, allowing the search range to adjust dynamically based on workload characteristics and observed accuracy.

4.4.1 Latency analysis: success vs. failure scenarios

The latency comparison in success vs. failure scenarios, shown in Fig. 13, provides insights into the system's performance. The graph distinguishes between *Success Latency* (green) and *Failure Latency* (red) across various percentiles (25th, 50th, 75th, 90th, and 99th), allowing a direct comparison of the time required in successful vs. unsuccessful search cases. This distinction reveals that failures typically incur higher latency, particularly at higher percentiles, indicating that when the sliding window search does not correct predictions within the specified range, the latency cost increases significantly.

Furthermore, this illustrates the *worst-case performance* (99th percentile) relative to typical cases (e.g., 50th percentile). Notably, the 99th percentile latency for failure cases is substantially higher, highlighting potential latency spikes in the most challenging scenarios. This observation is critical for understanding how the system performs under varying conditions.

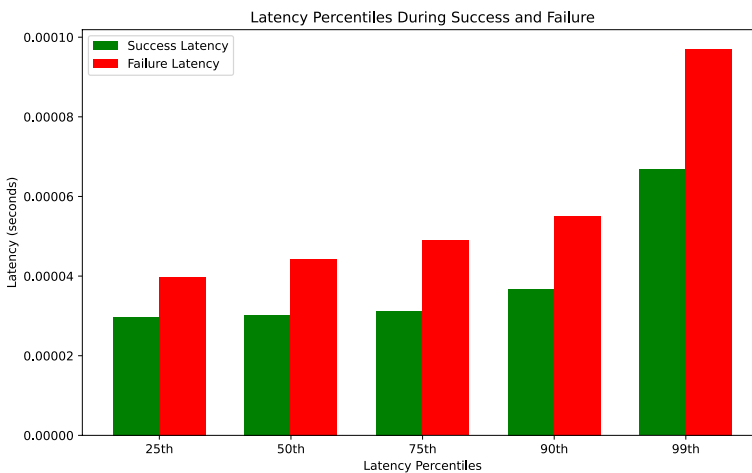


Fig. 13 Latency percentiles during success and failure scenarios for SWS-OGD

5 Conclusion

In this paper, a novel-learned indexing technique for blockchain querying was developed and evaluated. The results demonstrate that the proposed system offers a lower storage cost and a balanced performance across other key metrics.

This paper contributes to the field of blockchain technology by addressing the critical issue of storage efficiency in blockchain indexing. The learned indexing model developed provides a scalable and adaptable solution, paving the way for more efficient blockchain systems that can support a broader range of applications and enhance widespread adoption.

Despite its advantages, the proposed system has some limitations. One significant limitation is the CPU time required for updates, which can be further optimized. Additionally, the current system primarily focuses on point and range queries. Future work should explore optimizations for other queries, such as top-k queries, semantic searches, and more complex querying semantics and also using neural networks to improve prediction accuracy, to refine the indexing process, enabling even faster query responses. To ensure that such models remain practical for blockchain systems with resource constraints, techniques for reducing the size of neural networks could be investigated.

Funding Not applicable.

Data Availability The data used in this research will be made available upon request.

Code availability The code used for the experiments will be made available upon request.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Javaid M, Haleem A, Pratap Singh R, Khan S, Suman R (2021) Blockchain technology applications for industry 4.0: a literature-based review. *Blockchain Res Appl* 2(4):100027
2. Gad AG, Mosa DT, Abualigah L, Abohany AA (2022) Emerging trends in blockchain technology and applications: a review and outlook. *J King Saud Univ Comput Inf Sci* 34(9):6719–6742. <https://doi.org/10.1016/j.jksuci.2022.03.007>
3. Ali V, Norman AA, Azzuhri SRB (2023) Characteristics of blockchain and its relationship with trust. *IEEE Access* 11:15364–15374. <https://doi.org/10.1109/ACCESS.2023.3243700>
4. Wang J, Chen W, Wang L, Sherratt RS, Alfarraj O, Tolba A (2020) Data secure storage mechanism of sensor networks based on blockchain. *Comput Mater Continua* 65(3):2365–2384
5. Sunny J, Undralla N, Madhusudan Pillai V (2020) Supply chain transparency through blockchain-based traceability: an overview with demonstration. *Comput Ind Eng* 150:106895. <https://doi.org/10.1016/j.cie.2020.106895>
6. Zaabar B, Cheikhrouhou O, Jamil F, Ammi M, Abid M (2021) Healthblock: a secure blockchain-based healthcare data management system. *Comput Netw* 200:108500. <https://doi.org/10.1016/j.comnet.2021.108500>

7. Hewa TM, Hu Y, Liyanage M, Kanhare SS (2021) Yliantila M survey on blockchain-based smart contracts: technical aspects and future research. *IEEE Access* 9:87643–87662. <https://doi.org/10.1109/ACCESS.2021.3068178>
8. Ameyaw PD, Vries WT (2021) Toward smart land management: land acquisition and the associated challenges in ghana a look into a blockchain digital land registry for prospects. *Land*. <https://doi.org/10.3390/land10030239>
9. Musah S, Medeni TD, Soylu D (2019) Assessment of role of innovative technology through blockchain technology in ghana's cocoa beans food supply chains. In: 2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), pp. 1–12. <https://doi.org/10.1109/ISMSIT.2019.8932936>
10. Gyimah KN, Asiedu E, Antwi F (2023) Adoption of blockchain technology in the banking sector of ghana: opportunities and challenges. *Afr J Bus Manage* 17(2):32–42
11. Akraasi-Mensah NK, Tchao ET, Sikora A, Agbemenu AS, Nunoo-Mensah H, Ahmed A-R, Welte D, Keelson E (2022) An overview of technologies for improving storage efficiency in blockchain-based iiot applications. *Electronics*. <https://doi.org/10.3390/electronics11162513>
12. XiaoJu H, XueQing G, ZhiGang H, LiMei Z, Kun G (2020) Ebtree: A b-plus tree based index for ethereum blockchain data. In: Proceedings of the 2020 Asia Service Sciences and Software Engineering Conference. ASSE '20, pp. 83–90. Association for Computing Machinery, New York, NY, USA <https://doi.org/10.1145/3399871.3399892>
13. Jia D-Y, Xin J-C, Wang Z-Q, Lei H, Wang G-R (2021) Se-chain: a scalable storage and efficient retrieval model for blockchain. *J Comput Sci Technol* 36(3):693–706. <https://doi.org/10.1007/s11390-020-0158-2>
14. Zhu Y, Zhang Z, Jin C, Zhou A, Yan Y (2019) Sebdb: semantics empowered blockchain database. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 1820–1831 <https://doi.org/10.1109/ICDE.2019.00198>
15. Bitcoin blockchain size. https://ycharts.com/indicators/bitcoin_blockchain_size
16. Li Y, Zheng K, Yan Y, Liu Q, Zhou X (2017) Etherql: A query layer for blockchain system. In: Candan S, Chen L, Pedersen TB, Chang L, Hua W (eds) Database Systems for Advanced Applications. Springer, Cham, pp 556–567
17. Zhang Z, Zhong Y, Yu X (2021) Blockchain storage middleware based on external database. In: 2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP), pp. 1301–1304 <https://doi.org/10.1109/ICSP51882.2021.9408752>
18. Pratama, F.A., Mutijarsa, K.: Query support for data processing and analysis on ethereum blockchain. In: 2018 International Symposium on Electronics and Smart Devices (ISESD), pp. 1–5 (2018). <https://doi.org/10.1109/ISESD.2018.8605476>
19. Laishevskiy I, Barger A, Gorgadze V (2023) A journey towards the most efficient state database for hyperledger fabric. In: 2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1–3 <https://doi.org/10.1109/ICBC56567.2023.10174970>
20. Bragagnolo S, Marra M, Polito G, Gonzalez Boix E (2019) Towards scalable blockchain analysis. In: 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), pp. 1–7 <https://doi.org/10.1109/WETSEB.2019.00007>
21. El-Hindi M, Binnig C, Arasu A, Kossmann D, Ramamurthy R (2019) Blockchaindb: a shared database on blockchains. *Proc VLDB Endow* 12(11):1597–1609. <https://doi.org/10.14778/3342263.3342636>
22. Helmer S, Roggia M, Ioini NE, Pahl C (2018) Ethernitydb - integrating database functionality into a blockchain. In: Benczúr A, Thalheim B, Horváth T, Chiusano S, Cerquittelli T, Sidló C, Revesz PZ (eds) New trends in databases and information systems. Springer, Cham, pp 37–44
23. Sahoo MS, Baruah PK (2018) Hbasechaindb - a scalable blockchain framework on hadoop ecosystem. In: Yokota R, Wu W (eds) Supercomputing frontiers. Springer, Cham, pp 18–29
24. Abuhashim A, Tan CC (2020) Smart contract designs on blockchain applications. In: 2020 IEEE Symposium on Computers and Communications (ISCC), pp. 1–4 <https://doi.org/10.1109/ISCC50000.2020.9219622>
25. Thabet NA, Abdelbaki N (2021) Efficient quering blockchain applications. In: 2021 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES), pp. 365–369 <https://doi.org/10.1109/NILES53778.2021.9600533>
26. Gürsoy G, Brannon CM, Gerstein M (2020) Using ethereum blockchain to store and query pharmacogenomics data via smart contracts. *BMC Med Genomics* 13(1):74. <https://doi.org/10.1186/s12920-020-00732-x>

27. Chishti MS, Sufyan F, Banerjee A (2022) Decentralized on-chain data access via smart contracts in ethereum blockchain. *IEEE Trans Netw Serv Manage* 19(1):174–187. <https://doi.org/10.1109/TNSM.2021.3120912>
28. Han J, Seo Y, Lee S, Kim S, Son Y (2023) Design and implementation of enabling sql –query processing for ethereum-based blockchain systems. *Electronics*. <https://doi.org/10.3390/electronic s12204317>
29. Mardiansyah V, Muis A, Sari RF (2023) Multi-state merkle patricia trie (msmpt): high-performance data structures for multi-query processing based on lightweight blockchain. *IEEE Access* 11:117282–117296. <https://doi.org/10.1109/ACCESS.2023.3325748>
30. Huang T-L, Huang J (2022) An efficient storage structure and management for distributed ledgers in blockchain systems: an exploration based on purely theoretical approach. *IEEE Trans Netw Serv Manage* 19(4):3706–3723. <https://doi.org/10.1109/TNSM.2022.3195246>
31. Liu M, Wang H, Yang F (2021) An efficient data query method of blockchain based on index. In: 2021 7th International Conference on Computer and Communications (ICCC), pp. 1539–1544 <https://doi.org/10.1109/ICCC54389.2021.9674708>
32. Du P, Liu Y, Li Y, Yin H, Zhang L (2021) Etherh: A hybrid index to support blockchain data query. In: Proceedings of the ACM Turing Award Celebration Conference - China. ACM TURC '21, pp. 72–76. Association for Computing Machinery, New York, NY, USA <https://doi.org/10.1145/3472634.3472653>
33. Zeng L, Qiu W, Wang X, Wang H, Yao Y, Yu Z (2021) Transaction-based static indexing method to improve the efficiency of query on the blockchain. In: 2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), pp. 780–784 <https://doi.org/10.1109/ICAICA52286.2021.9497966>
34. Wan L (2021) A query optimization method of blockchain electronic transaction based on group account. In: Atiquzzaman M, Yen N, Xu Z (eds) Big data analytics for cyber-physical system in smart city. Springer, Singapore, pp 1358–1364
35. Pei Q, Zhou E, Xiao Y, Zhang D, Zhao D (2020) An efficient query scheme for hybrid storage blockchains based on merkle semantic trie. In: 2020 International Symposium on Reliable Distributed Systems (SRDS), pp. 51–60 <https://doi.org/10.1109/SRDS51746.2020.00013>
36. Ruan P, Dinh TTA, Lin Q, Zhang M, Chen G, Ooi BC (2021) Lineagechain: a fine-grained, secure and efficient data provenance system for blockchains. *VLDB J* 30(1):3–24. <https://doi.org/10.1007/s00778-020-00646-1>
37. Zhu Y, Zhang Z, Jin C, Zhou A, Yan Y (2019) Sebdb: Semantics empowered blockchain database. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 1820–1831 <https://doi.org/10.1109/ICDE.2019.00198>
38. Xing X, Chen Y, Li T, Xin Y, Sun H (2021) A blockchain index structure based on subchain query. *J Cloud Comput* 10(1):52. <https://doi.org/10.1186/s13677-021-00268-0>
39. Xu C, Zhang C, Xu J (2019) vchain: Enabling verifiable boolean range queries over blockchain databases. In: Proceedings of the 2019 International Conference on Management of Data. SIGMOD '19, pp. 141–158. Association for Computing Machinery, New York, NY, USA <https://doi.org/10.1145/3299869.3300083>
40. Hao K, Xin J, Wang Z, Yao Z, Wang G (2022) On efficient top-k transaction path query processing in blockchain database. *Data Knowl Eng* 141:102079. <https://doi.org/10.1016/j.datak.2022.102079>
41. Kraska T, Beutel A, Chi EH, Dean J, Polyzotis N (2018) The case for learned index structures. In: Proceedings of the 2018 International Conference on Management of Data. SIGMOD '18, pp. 489–504. Association for Computing Machinery, New York, NY, USA <https://doi.org/10.1145/3183713.3196909>
42. Ding J, Minhas UF, Yu J, Wang C, Do J, Li Y, Zhang H, Chandramouli B, Gehrke J, Kossmann D, Lomet D, Kraska T (2020) Alex: An updatable adaptive learned index. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. SIGMOD '20, pp. 969–984. Association for Computing Machinery, New York, NY, USA <https://doi.org/10.1145/3318464.3389711>
43. Ge J, Zhang H, Shi B, Luo Y, Guo Y, Chai Y, Chen Y, Pan A (2023) Sali: a scalable adaptive learned index framework based on probability models. *Proc ACM Manag Data*. <https://doi.org/10.1145/3626752>
44. Zhang C, Xu C, Hu H, Xu J (2024) Cole: A column-based learned storage for blockchain systems (technical report)

45. Yao Z, Xin J, Hao K, Wang Z, Zhu W (2023) Learned-index-based semantic keyword query on blockchain. *Mathematics*. <https://doi.org/10.3390/math11092055>
46. Chang J, Li B, Xiao J, Lin L, Jin H (2023) Anole: a lightweight and verifiable learned-based index for time range query on blockchain systems. In: Wang X, Sapino ML, Han W-S, El Abbadi A, Dobbie G, Feng Z, Shao Y, Yin H (eds) *Database systems for advanced applications*. Springer, Cham, pp 519–534
47. Hoi SCH, Sahoo D, Lu J, Zhao P (2021) Online learning: a comprehensive survey. *Neurocomputing* 459:249–289. <https://doi.org/10.1016/j.neucom.2021.04.112>
48. Zhang J, Sun Y, Guo D, Luo L, Li L, Nian Q, Zhu S, Yang F (2024) A reputation awareness randomization consensus mechanism in blockchain systems. *IEEE Internet Things J* 11(20):32745–32758. <https://doi.org/10.1109/JIOT.2024.3408846>
49. Hoi SCH, Sahoo D, Lu J, Zhao P (2021) Online learning: a comprehensive survey. *Neurocomputing* 459:249–289. <https://doi.org/10.1016/j.neucom.2021.04.112>

Authors and Affiliations

Emmanuel Acheampong Asiamah^{1,2} · Nana Kwadwo Akrasi-Mensah^{1,2} · Prince Odame^{1,2} · Eliel Keelson^{1,2} · Andrew Selasi Agbemenu^{1,2} · Eric Tutu Tchao^{1,2} · Mohammed Al-Khalidi³ · Griffith Selorm Klogo^{1,2}

✉ Emmanuel Acheampong Asiamah
eaasiamah4@st.knust.edu.gh

Nana Kwadwo Akrasi-Mensah
nkakrasimensah3@st.knust.edu.gh

Prince Odame
podame.coe@knust.edu.gh

Eliel Keelson
ekeelson@knust.edu.gh

Andrew Selasi Agbemenu
asagbemenu@knust.edu.gh

Eric Tutu Tchao
ettchao.coe@knust.edu.gh

Mohammed Al-Khalidi
m.al-khalidi@mmu.ac.uk

Griffith Selorm Klogo
gsklogo.coe@knust.edu.gh

¹ Distributed IoT Platforms, Privacy and Edge-Intelligence Research (DIPPER) Laboratory, Faculty of Electrical and Computer Engineering, Kwame Nkrumah University of Science and Technology, PMB, Kumasi, Ashanti, Ghana

² Department of Computer Engineering, Kwame Nkrumah University of Science and Technology, PMB, Kumasi, Ashanti, Ghana

³ Department of Computing and Mathematics, Manchester Metropolitan University, Manchester M15 6BH, UK