Article | Received 28 October 2025; Revised 1 February 2026; Accepted 9 February 2026; Published 26 February 2026
https://doi.org/10.55092/blockchain20260002

Optimizing Bitcoin transaction validation with segregated witness: a single-validation approach

Shunqing Xu, Yanqi Xu, Dawei Nie, Tianang Yao, Lili Yao, Huayao Wu, Changhai Nie* and Chen Tian

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

* Correspondence author; E-mail: changhainie@nju.edu.cn.

Highlights:

- Proposes a deterministic one-time validation scheme via SegWit.
- Utilizes wtxid indexing to eliminate redundant verifications.
- Reduces total transaction validation overhead by approximately 50%.

Abstract: The Bitcoin system has operated reliably for over 16 years, attracting considerable attention due to its decentralized architecture and the inherent value of its digital assets. Nevertheless, performance limitations, especially in transaction validation, continue to impede its scalability and efficiency. In the Bitcoin network, each full node not only needs to validate a transaction when it is first received, but also needs to validate it again when it is packaged into a new block. This redundant validation mechanism reduces the overall efficiency of the Bitcoin system. Utilizing Segregated Witness technology, this paper introduces a one-time validation optimization scheme. Specifically, we leverage the Witness Transaction ID (wtxid) inherent in SegWit to build a deterministic index mapping between the local transaction pool and new candidate blocks. By checking this index, the system can identify and skip the redundant verification of transactions that have already been validated upon entry into the pool. The proposed approach eliminates the need for secondary validation when processing new blocks, specifically for transactions already stored in the local transaction pool. Through comprehensive theoretical analysis, we demonstrate that this optimization can reduce the transaction validation overhead by approximately 50% without compromising the security of the Bitcoin network. In short, our research provides an innovative solution to Bitcoin's performance challenges, thereby contributing to its future development and long-term sustainability.

Keywords: blockchain; Bitcoin; segregated witness; transaction validation

1. Introduction

Since the official launch of the main Bitcoin [1] network in 2009, the system has maintained stable operations for over 16 years. Due to its anonymity and valuable digital asset attributes, Bitcoin has

attracted significant attention from both academia and industry. To ensure the security of the Bitcoin system, full nodes are responsible for validating new transactions. Only transactions that can pass validation are retained in the local transaction pool and subsequently propagated to other nodes. Over time, as a full node continues to operate, most transactions included in newly received blocks are already present in its local transaction pool. A single block often contains numerous transactions, each consisting of multiple inputs (excluding the coinbase transaction). Consequently, validating these transactions consumes a significant portion of the total block processing time. As of December 19, 2025, there have been 1,284,669,135 Bitcoin transactions (https://www.blockchain.com/explorer/mempool/btc) recorded on the blockchain. Under the current operational mechanics of the Bitcoin system, transactions are compelled to undergo multiple rounds of validation. This redundancy significantly reduces the efficiency of block validation and imposes a substantial performance overhead on the network.

Besides, inefficiencies in block validation can pose a potential threat to the security of the Bitcoin system. It is well established that a full node disseminates a new block to its neighboring nodes only after successfully completing the block validation process. Slow block validation results in prolonged propagation delay, which will increase the risk of blockchain forks and lead to data inconsistencies across different nodes.

In response to the low throughput rate and high confirmation delay issues affecting the Bitcoin system, numerous researchers have conducted extensive investigations in recent years [2–6]. For example, Yang *et al.* [7] proposed a stateless transaction validation scheme utilizing dual RSA accumulators. By employing fixed-size cryptographic commitments instead of stateful data, this approach significantly reduces local storage requirements while supporting independent transaction validation. Erdin *et al.* [8] introduced a decentralized payment network utilizing the Lightning Network. This approach addresses high transaction fees and lengthy validation times, enabling Bitcoin to handle larger transaction volumes. Dai *et al.* [9] proposed the Efficient Block Validation (EBV) mechanism, which di-vides transaction input checking into three components: presence validation, unspent validation, and script vali-dation, effectively accelerating the transaction input validation process. Using a simulator and real network data, Kedziora *et al.* [10] compared block propagation times before and after Segregated Witness implementation. Their results revealed a significant reduction from 12.5 seconds to 0.68 seconds on average. These findings validate Segregated Witness's performance enhancements and provide a theoretical basis for its optimizations. Kim *et al.* [11] introduced the Demand-Aware Pre-filled Compact Block Relay (DAP-CBR) scheme, which improves block propagation efficiency in the Bitcoin network by dynamically pre-filling transactions. This method is compatible with Segregated Witness and can reduce block validation time by 39.1%.

Beyond the classic optimization strategies, recent academic research has explored diverse approaches to address the scalability bottlenecks of blockchain systems. These works provide a broader context for our proposed single-validation scheme. First, parallel transaction execution has emerged as a promising di-rection. Anjana *et al.* [12] proposed a parallel execution methodology leveraging conflict specifications. They achieved significant throughput improvements by identifying non-conflicting transactions suitable for simultaneous processing on multicore architectures. This highlights the industry's shift from serial to parallel validation paradigms. Second, optimizing the network layer remains a critical focus. Recent work [13] introduced an Age-of-Information (AoI) based framework to optimize block propagation efficiency in Web 3.0 networks. By modeling the

freshness of blocks and designing incentive mechanisms for miners, this approach effectively reduces propagation latency, which aligns with our goal of accelerating block confirmation. Furthermore, stateless validation techniques have gained traction. Innovations such as SecChain utilize novel state commitment schemes to enable secure stateless sharded blockchains [14]. These protocols allow nodes to validate transactions without maintaining the full ledger state, thereby reducing storage overhead.

Despite significant progress, existing solutions either introduce considerable system complexity or fail to ad-dress the issue of redundant verification for transactions already stored in the local transaction pool. This paper optimizes the transaction verification process from the perspective of Bitcoin's full life cycle, proposing a one-time verification optimization scheme based on Segregated Witness. The proposed optimization eliminates the need for secondary validation of transactions when processing a new block. We conduct a comprehensive theoretical analysis of the proposed method, demonstrating a massive improvement in transaction validation efficiency without compromising the security of the validation process. For the remainder of this paper, the term "SegWit" will be used to refer to Segregated Witness.

The main contributions of this paper are summarized as follows:

(1) We propose a novel, deterministic one-time validation scheme leveraging SegWit. Unlike existing probabilistic cache mechanisms that may suffer from eviction, our approach utilizes the distinct wtxid index in the local Mempool to explicitly and reliably eliminate redundant checks for known transactions.

(2) We implement an optimized validation algorithm with $O(1)$ time complexity for transaction existence checks. Theoretical analysis and simulation results demonstrate that our method reduces the overall trans-action validation overhead by up to 50% throughout the transaction lifecycle, significantly enhancing block processing efficiency.

(3) We establish a formal security model predicated on the collision resistance of cryptographic hash functions. We provide a systematic proof demonstrating that our scheme maintains validation equivalence with the traditional dual-validation protocol, ensuring robust security against double-spending and malicious block attacks.

It is also worth acknowledging that the current reference implementation, Bitcoin Core, employs internal optimization mechanisms such as ValidationCache to mitigate verification overhead. We will examine these existing probabilistic mechanisms in detail and contrast them with our proposed deterministic approach in the subsequent sections.

The remainder of the paper is structured as follows: Section 2 presents preliminary knowledge, analyzing the data structure of transactions, SegWit, the life cycle of Bitcoin transactions and blocks, and reviewing the existing optimization mechanisms in Bitcoin Core. Section 3 details the proposed optimization scheme, including the formal definitions of transaction validation, the specific algorithmic design, and a complexity analysis. Section 4 delves into the theoretical analysis, establishing a formal security model and evaluating both the validation efficiency and security of our proposal. Section 5 discusses the comparative advantages of our deterministic scheme over existing probabilistic caching mechanisms and incorporates technical insights from the community. Finally, Section 6 concludes the paper and outlines avenues for future research.

2. Preliminary

This section outlines the preliminaries, covering Bit-coin transaction structures, SegWit mechanics, and the validation lifecycle. Furthermore, we examine the existing probabilistic caching mechanisms in Bitcoin Core, highlighting their limitations to establish the context for our deterministic optimization.

A. Transaction Structure

A transaction (often abbreviated as tx) represents an exchange of value between two or more parties by monetary means, essentially indicating a transfer of ownership. Similar to conventional bank transfers, Bitcoin transactions involve the movement of funds. They are characterized by specific details, including the source (input address), destination (output address), and transfer amount. This process facilitates the creation and transfer of digital assets. It also enables tracking the complete history of each Bitcoin flow.

A transaction comprises two primary components: transaction input and transaction output, as illustrated in Figure 1. In a transaction, Bitcoins from one or more accounts can be utilized as inputs and transferred to one or more other accounts. Additionally, each transaction includes essential metadata such as the current version of the transaction software and the lock time used to confirm the transaction within the block.

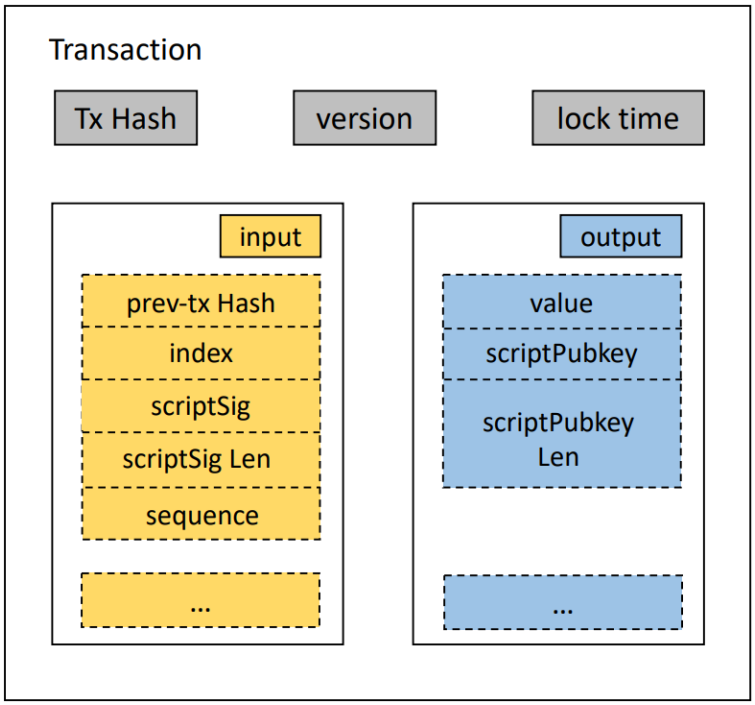

Figure 1. The data structure of Bitcoin transaction.

The input of each transaction primarily includes the hash of the previous transaction (prev-tx Hash), the output index of the previous transaction (index), the input script (scriptSig), the length of the input script (scriptSig Len), and the sequence of transaction inputs. The output of each transaction mainly comprises the transfer amount (value), the output script (scriptPub-Key), and the length of the output script (scriptPubkey Len). The unlocking script (input script) is synchronized with the locking script (output script) in the referenced output to verify the validity of the current

transaction. When the unlocking script satisfies the conditions of the locking script, the input is considered valid.

B. SegWit

SegWit, introduced via Bitcoin Improvement Propos-als (BIPs) 141–144 and 145 [15–19], is a protocol upgrade activated in 2017 to address fundamental limitations in Bitcoin's transaction structure and scalability. At its core, SegWit separates (or "segregates") the witness data—primarily the signatures (scriptSig) that prove transaction authorization—from the main transaction body. In legacy (non-SegWit) transactions, signatures are embedded within the input fields, contributing to the transaction ID (txid), which is a double SHA-256 hash of the entire transaction. SegWit relocates this witness data to a new field outside the main body, marked by a flag, as illustrated in Figure 2. This results in two distinct identifiers:

- txid (transaction ID): A hash of the non-witness data (e.g., inputs, outputs, version, and lock time), ensuring backward compatibility with non-upgraded nodes.
- wtxid (witness transaction ID): A hash that includes the witness data, providing a unique, im-mutable commitment to the full transaction

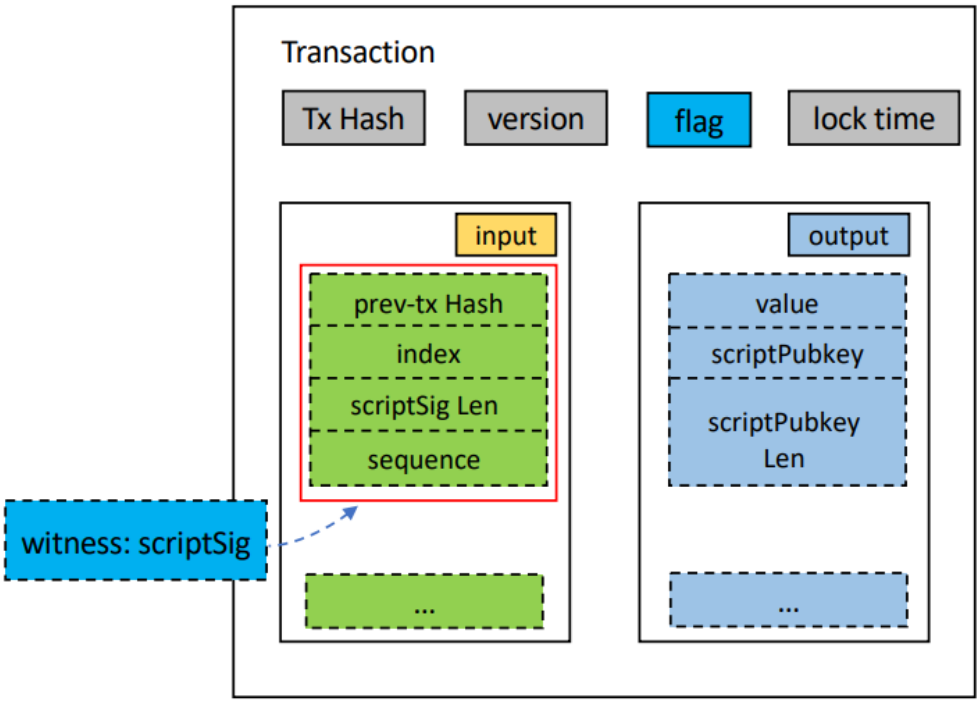

Figure 2. The data structure of a transaction with SegWit.

Transaction malleability allows third parties to modify signatures without invalidating the transaction. This modification alters the txid while preserving the trans-action's underlying validity [20]. This complicates the implementation of second-layer networks. Furthermore, applications relying on unconfirmed transactions may fail when processing differing txids for the same trans-action, potentially leading to errors or double-spending attacks. SegWit technology effectively separates signature data, ensuring the uniqueness and tamper-resistance of the wtxid. This addresses the unpredictability of transaction IDs caused by malleability, preventing disruptions to dependent transactions, such as those in multi-signature scenarios or Lightning Network channels. Additionally,

SegWit introduces a “block weight” metric, replacing the legacy 1 MB size limit with a 4 million weight unit (WU) cap. This effectively increases the block capacity for transactions without requiring a hard fork. With more and more full nodes supporting SegWit, this technology will undoubtedly become the mainstream direction in the future.

C. Bitcoin Transaction Life Cycle

Fundamentally, a Bitcoin transaction involves the transfer of ownership through a multi-stage process. When a client initiates a transfer, the transaction is first constructed and propagated across the P2P network. Upon receiving this unconfirmed transaction, a full node performs an initial validity check—verifying inputs, signatures, and format—before adding it to its local transaction pool (Mempool). This constitutes the first validation phase. Once validated, the transaction waits in the Mempool until a miner selects it for inclusion in a new candidate block. The lifecycle concludes when the block is mined, propagated, and confirmed by the network, achieving finality after sufficient subsequent blocks are appended.

D. New Block Life Cycle

The lifecycle of a block begins when a miner successfully solves the Proof-of-Work puzzle, packaging a set of transactions from their Mempool into a new block. This block is then broadcast to neighboring nodes. Crucially, upon receiving a new block, every full node must independently verify its legitimacy. This process involves checking the block header (PoW validity) and, more importantly, re-validating every trans-action contained within the block body to ensure no double-spending or illicit inputs. This step constitutes the second validation phase. Only after this rigorous verification is the block appended to the local blockchain and propagated further. As noted, this mechanism forces nodes to re-verify transactions that they likely already validated when they first entered the Mempool.

E. Existing Optimization Mechanisms in Bitcoin Core

Before introducing our proposed scheme, it is essential to review the existing optimization mechanisms within the reference implementation of Bitcoin (Bitcoin Core) (https://github.com/Bitcoin/Bitcoin/blob/master/src/validation.cpp). To mitigate the CPU overhead of repeated script verifications, Bitcoin Core implements a CuckooCache-based mechanism, managed primarily through the ValidationCache class. As presented in Listing 1, the Check-InputScripts function prioritizes querying the verification cache to retrieve existing results before attempting full execution.

(1) Cache Hit: If it contains true, indicating that the script execution has already been verified and cached. In this case, the system immediately re-turns true, skipping the computationally expensive script evaluation.

(2) Cache Miss: If the entry is not found, the system proceeds to execute the full validation logic.

However, it is crucial to note that these caches are memory-bounded. As shown in Listing 1, the Valida-tionCache constructor explicitly initializes the cache size by calling *m_script_execution_cache.setup_bytes* with a fixed limit (*script_execution_cache_bytes*). This initialization logic confirms that the cache operates under strict memory constraints. Consequently, under high network load, valid entries are subject to eviction policies. Consider a transaction that was verified upon entering the Mempool. If its cache entry is evicted to accommodate newer data, this transaction requires full re-validation when it subsequently appears in a block.

```
// Class: ValidationCache
// Location: src/validation.cpp

// Validation Logic.
bool CheckInputScripts(const CTransaction& tx, ...) {
        // 1. Compute Cache Key (Hash + Flags)
        // The hasher combines witness hash and verify flags
        uint256 hashCacheEntry;
        CSHA256 hasher = validation_cache. ScriptExecutionCacheHasher();
        hasher.Write(UCharCast(tx. GetWitnessHash().begin()), 32)
                .Write((unsigned char*)&flags, sizeof(flags))
                .Finalize(hashCacheEntry.begin());

        // 2. Cache Lookup
        // If found, skip redundant verification
        if (validation_cache. m_script_execution_cache.contains(hashCacheEntry, ! cacheFullScriptStore)) {
                return true; // Cache Hit
        }

        // ... (Perform full signature verification loop)...

        // 3. Cache Insertion
        // Store result if validation succeeds
        if (cacheFullScriptStore && !pvChecks)
        {
                validation_cache. m_script_execution_cache.insert (hashCacheEntry);
        }
        return true;
}

// Initialization Logic.
ValidationCache::ValidationCache(size_t script_execution_cache_bytes, ...){
        // ... (Hasher setup omitted) ...

        // Memory Initialization
        // 'setup_bytes' enforces the fixed memory limit
        const auto [num_elems, approx_size_bytes]=
                m_script_execution_cache. setup_bytes(script_execution_cache_bytes);
}
```

Listing 1. Simplified representation of the existing validation logic in Bitcoin Core. The system first checks the *ValidationCache*. If the transaction hash exists (Cache Hit), verification is skipped. Note that the cache is initialized with a fixed memory limit (*setup_bytes*), implying that valid entries may be evicted under high load.

3. Optimization scheme

A. Definition of Transaction Validation

Based on the lifecycle analysis in Section 2, it is evident that a Bitcoin transaction undergoes validation at two distinct phases. First, an initial check occurs during the transaction's propagation across the network. Second, a repetitive check is performed when the trans-action is packaged into a new candidate block. To clearly differentiate these verification contexts and quantify the redundancy, we formally define these two validation instances as follows.

Definition 1. First transaction validation. In the life cycle of a Bitcoin transaction, when a full node initially receives a new transaction from the P2P network, it performs a comprehensive validity check. Only after passing this check is the transaction accepted into the local memory pool (Mempool) [21–23]. This process, which verifies consensus compliance (e.g., syntax correctness, signature validity, and input existence), is referred to as the first transaction validation.

Definition 2. Second transaction validation. In the life cycle of a block, when a full node receives a new block from the network, it performs the validity check on every transaction within the block body. This is referred to as the second transaction validation.

This process is critical, as it ensures that miners cannot submit illicit blocks by independently verifying the integrity of every transaction. Analysis of the Bitcoin source code (https://github.com/Bitcoin/Bitcoin) reveals that the validation criteria for this second check are identical to those applied during the first transaction validation. As depicted in Part 1 of Listing 2, when a transaction is propagated to a full node, it undergoes the first validation (via MemPoolAccept::PreChecks). Subsequently, when the transaction is included in a new block and propagated to other nodes, each node performs the second validation as shown in Part 2 of Listing 2 (CheckBlock), which redundantly invokes the same verification logic. Consequently, this redundancy presents a significant opportunity for optimization, particularly given that the time expended on validating transactions within a block constitutes a substantial portion of the overall block validation duration [16]. Theoretically, this suggests that secondary validation could be omitted for transactions already stored in the local transaction pool, potentially saving considerable time for miners and enhancing the performance of the Bitcoin system as a whole. However, the implications for the security of the Bitcoin network warrant further detailed investigation to ensure that such an optimization does not introduce vulnerabilities.

```
// Part 1: First Validation (Mempool Entry)
// Location: src/validation.cpp
bool MemPoolAccept::PreChecks(ATMPArgs& args, Workspace& ws) {
        const CTransaction& tx = *ws.m_ptx;
        TxValidationState& state = ws.m_state;

        // A. Context-free checks (Syntax, Size, Format)
        // CRITICAL: This exact function is called again in CheckBlock
        if (!CheckTransaction(tx, state)) {
                return false; // Validation failed
        }

        // ... (Context-dependent checks: Inputs existence, Coinbase rules)...

        // B. Check for conflicts with in-memory transactions
        if (m_pool.exists(tx.GetWitnessHash()))
        {
                return state.Invalid(TxValidationResult::TX_CONFLICT, "txn-already-in-mempool");
        }

        return true;
}

// Part 2: Second Validation (New Block Received)
// Location: src/validation.cpp
bool CheckBlock(const CBlock&block, BlockValidationState& state, ...) {
        // ... (Header and Merkle Root checks)...

        // Iterate through all transactions in the new block
        for (const auto& tx : block.vtx) {
                TxValidationState tx_state;

                // REDUNDANT VALIDATION:
                // The block validator re-executes CheckTransaction for every tx,
                // repeating the exact checks performed in PreChecks.
                if (!CheckTransaction(*tx, tx_state)) {
                        return state.Invalid( BlockValidationResult::BLOCK_CONSENSUS, tx_state. GetRejectReason());
                }
        }

        return true;
}
}
```

Listing 2. Comparison of validation logic between Mempool acceptance (First Validation) and Block processing (Second Validation). Both processes invoke the identical CheckTransaction function to verify consensus rules (e.g., syntax, duplicates), proving the existence of redundant validation.

B. Algorithmic Design Strategy

To systematically implement the proposed one-time validation scheme, we designed an optimized block validation algorithm tailored for SegWit-enabled nodes. The core logic shifts from a "verify-all" approach to a "verify-by-exception" approach, leveraging the deterministic index of the local memory pool (Mempool).

The core validation logic is formally described in Algorithm 1 and visually abstracted in Figure 3. Unlike the traditional verification mechanism, our strategy adopts a "verify-by-exception" approach. Specifically, upon receiving a new candidate block, the node initiates an iteration through the transaction list. For each entry, the algorithm computes its unique Witness Transaction ID (wtxid) and utilizes this identifier as a deterministic key to query the local Mempool. This lookup mechanism allows the system to identify validated transactions immediately and skip redundant cryptographic checks.

Algorithm 1 Optimized Block Validation Strategy

Require:
B: Candidate Block containing transactions $\{tx_1, tx_2, ..., tx_n\}$
M: Local Mempool containing validated transaction indices
Ensure:
True if the block is valid, **False** otherwise

```
function OPTIMIZEBLOCKVALIDATION(B, M)
// Step 1: Basic Block Header Validation
if ¬ CHECKBLOCKHEADER(B) then
return False
end if
// Step 2: Iterate through transactions in the block
for each tx in B.transactions do
    wtxid ← CALCULATEWTXID(tx)
    // Optimized Path: Check existence in Mem-pool
    if M.contains(wtxid) then
        // Skip redundant signature/script checks
        continue
    else
        // Fallback Path: Perform full validation
        if ¬ STANDARDTRANSAC-TIONCHECK(tx) then
            return False
        end if
    end if
end for
return True
end function
```

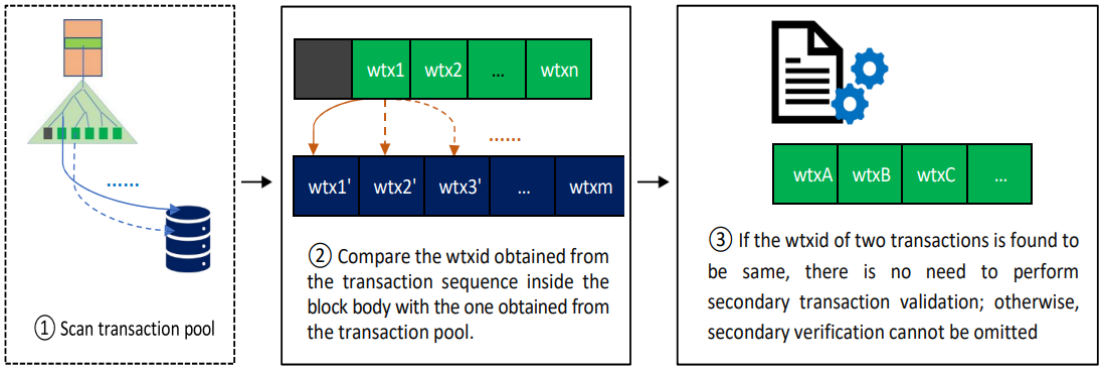

Figure 3. The optimization scheme.

C. Operational Workflow Analysis

This section details the step-by-step execution flow of the proposed single-validation mechanism when a full node receives a new candidate block. The process is divided into five critical stages:

Step 1: Header Validation. Upon receiving a new block, the node first validates the block header. This includes checking the Proof-of-Work (PoW) nonce, the timestamp, and the Merkle root. This step is identical to the standard Bitcoin protocol and acts as a preliminary filter to reject obviously invalid blocks immediately.

Step 2: Transaction Iteration and Identifier Ex-traction. The system initiates an iteration through the transaction list contained in the block body. For each transaction tx_i, the system calculates its unique identifier. In our SegWit-based scheme, we specifically compute the *wtxid*, which covers both the transaction data and the witness data, ensuring no part of the transaction has been tampered with.

Step 3: Deterministic Index Lookup. Unlike the probabilistic *ValidationCache* in Bitcoin Core (which may evict entries under memory pressure), our scheme queries the local Transaction Pool (Mempool) Index. This lookup determines whether tx_i currently resides in the local Mempool.

Step 4: Conditional Verification Logic. Based on the lookup result, the workflow branches:

- Case A (Hit-Optimized Path): If the *wtxid* is found in the index, it implies tx_i has already passed the "First Validation" (consensus rules and script checks) upon entering the pool. The node immediately marks tx_i as valid, bypassing the signature verification and script execution. This is the primary source of efficiency gain.
- Case B (Miss-Fallback Path): If the *wtxid* is not found, the system seamlessly falls back to the standard validation procedure. The node performs the full CheckTransaction routine to ensure security.

Step 5: State Update. Once all transactions in the block are processed, if no invalid transactions are found, the block is accepted into the local blockchain. Concurrently, the Mempool updates the confirmed transactions to maintain consistency.

D. Complexity Analysis

We formally analyze the computational and spatial complexity of the proposed scheme to demonstrate its efficiency and scalability.

(1) Time Complexity: Let N be the number of trans-actions in a block, and T_{sig} be the average time required to verify digital signatures and scripts for a single trans-action. In the traditional approach, the time complexity for validating a block is $O(N \cdot T_{sig})$. Since cryptographic operations are CPU-intensive, T_{sig} is significant. In our optimized approach, let T_{hash} be the time to compute a wtxid and perform a hash map lookup. Since the Mempool uses a hash index, the lookup operation is $O(1)$. Thus, for a block where k transactions are already in the Mempool (Hit) and $N-k$ are new (Miss), the total time is:

$$\mathrm{Toptimized} = \mathrm{k} \cdot \mathrm{Thash} + (\mathrm{N}-\mathrm{k}) \cdot \mathrm{Tsig} \quad (1)$$

Given that $T_{hash} \ll T_{sig}$ and, in a synchronized network, $k \approx N$ (most transactions are known), the complexity approaches $O(N \cdot O(1))$, representing a linear reduction in validation overhead proportional to the Mempool hit rate.

(2) Space Complexity: The proposed scheme lever-ages the existing *CTxMemPool* data structure in Bit-coin Core, which already maintains a *wtxid* index for transaction management. Therefore, our

optimization requires $O(1)$ additional space overhead, as it does not necessitate constructing new auxiliary large-scale data structures or caching tables. This makes it highly memory-efficient compared to independent cache lookaside buffers.

4. Theoretical analysis

A. Security Model

To rigorously analyze the security of our proposed optimization, we define the following security model:

Cryptographic Primitives: We assume that the under-lying cryptographic hash functions (e.g., SHA-256 used in Bitcoin) satisfy the resistance to collision. Specifically, it is computationally infeasible for an adversary to find two distinct transactions Tx_A and Tx_B such that $wtxid(Tx_A) = wtxid(Tx_B)$.

Adversary: We consider an adversary who acts as a full node or as a miner. The adversary can broadcast arbitrary transactions and blocks to the network. The adversary aims to deceive the victim node into accepting an invalid transaction by exploiting the skipped validation step.

Security Goal: Our goal is to ensure validation equivalence. The outcome of our optimized validation process must be identical to the traditional full validation process. An invalid transaction included in a block must never be accepted, even if a hash collision is attempted against the local memory pool.

B. Efficiency Analysis of Validation

In this part, we focus on analyzing how much the optimization scheme can improve the validation efficiency of transactions using SegWit. Suppose in the traditional Bitcoin P2P network, node A broadcasts a new transaction to its neighboring node B. When B receives the transaction, it needs to perform the first validation, which requires t_1 time on average. After that, when B receives a new block, for every transaction in it, B also needs t_1 time to perform the second validation. Therefore, the total validation time for a traditional Bitcoin transaction is T_1, given by:

$$T_1 = 2t_1 \tag{2}$$

For transactions using our optimization scheme, the second validation does not need to perform CheckTransaction operation defined in the Bitcoin source code, and only needs to find whether its wtxid has an index item in the transaction pool, which we denote by t_2 time. Note that Bitcoin already uses multi-index hashes to store entries in the transaction pool, so this does not introduce additional space overhead. Under our optimization, each transaction only needs to be validated once. Therefore, the total validation time for a transaction using the optimized scheme is T_2, given by:

$$T_2 = t_1 + t_2 \tag{3}$$

According to the above two formulas, we can calculate the time improvement rate of the optimal solution. We denote it by R, given by:

$$R = \frac{T1 - T2}{T1} \tag{4}$$

From the formula, it is evident that when t_1 approaches infinity (or becomes sufficiently large), the impact of t_2 becomes negligible. Consequently, the time improvement rate R for computing the optimal solution ranges from 0% to 50%.

To empirically evaluate the efficiency improvements, we conducted experiments on a machine running Windows 11 (Version 22H2). The hardware configuration consisted of an Intel Core i9-13900HX CPU @ 2.20 GHz and 16 GB of RAM. The simulation framework was implemented in standard C++17 and compiled using the GCC compiler with standard optimization enabled. Additionally, we utilized the OpenSSL library to handle cryptographic operations, ensuring performance consistency with standard Bitcoin implementations.

We simulate the conventional Bitcoin transaction validation process based on the Bitcoin source code, which checks the transaction's basic structure, serialized size, duplicate input, and so on. Then, we implement the proposed optimization by maintaining a local transaction pool containing 125,000 transactions. We compare the time consumption before and after optimization; the experimental results are shown in Table 1.

Table 1. validation time consumed.

Number of tx	T_1[1]	T_2[1]	R
500	6575.85	3288.27	49.99%
1000	12031	6016.09	49.99%
1500	15376.7	7689.26	49.99%
2000	22484.2	11243.4	49.99%
2500	28440.7	14222	49.99%
3000	34071.8	17037.7	49.99%
3500	42351.3	21177.8	50%

[1] In milliseconds.

In our simulation, the transaction count per block was scaled from 500 to 3500. While we acknowledge that recent network activity has occasionally produced blocks containing over 4000 transactions (e.g., due to specific transaction structures or SegWit adoption), public blockchain data indicates that the average transaction count per block typically ranges from 1500 to 3000 (https://www.blockchain.com/explorer/blocks/btc). Consequently, selecting 3500 transactions as the up-per bound provides a rigorous stress test that exceeds typical network averages. Moreover, since our proposed validation mechanism exhibits linear time complexity ($O(N)$) with constant-time ($O(1)$) lookups, the efficiency gains verified at 3500 transactions are theoretically consistent and extensible to larger block sizes. Empirical results indicate that our optimization reduces the total validation overhead by approximately 50% compared to the traditional approach. It is important to emphasize that the values of R in Table 1 are empirical results, not theoretical limits. We did not measure T_{hash} in isolation because it is a fast $O(1)$ in-memory lookup. Its cost is negligible compared to the resource-intensive signature verification T_{sig}. The fact that R converges to 50% validates our premise. It demonstrates that the overhead of T_{hash} has almost no impact on the total validation time.

C. Security Analysis

In this section, we conduct a comprehensive security analysis of our proposed optimization, assessing the feasibility of potential attacks, the acceptability of SegWit, and the overall security of the

Bitcoin network under the proposed optimization. Notably, we shared our research ideas with the community, receiving valuable support and positive feedback. Through this study, we aimed to determine whether this optimization would compromise the security guarantees inherent in Bitcoin.

(1) Potential Double-Spending Concern: The proposed scheme, which optimizes the second validation step for transactions already present in the Mem-pool, does not introduce double-spending vulnerabilities within the Bitcoin network. This is mainly due to Bitcoin's powerful memory pool management mechanism. If a transaction in the Mempool shares inputs with another transaction included in the newly accepted block, the conflicting transaction in the Mempool will be automatically removed or marked as invalid. This ensures that any attempt to double-spending—where two transactions try to spend the same UTXO—is effectively neutralized.

(2) Modification of External Transaction Pools: Bitcoin operates on a decentralized network composed of numerous independent full nodes, each maintaining its own transaction pool. These pools temporarily store transactions before they are included in a block by miners. The decentralized nature implies that no single entity controls all transaction pools, making coordinated attacks across multiple nodes exceedingly challenging. For an attacker to successfully modify transactions within others' Mempools, they would need to simultaneously gain control over a significant proportion of the network's nodes. Given the vast distribution and het-erogeneity of Bitcoin nodes, achieving this level of control would require an impractical amount of resources and coordination. Additionally, each node independently validates transactions before including them in its Mem-pool, ensuring that only valid transactions can be widely disseminated on the network. The economic cost and logistical complexity associated with orchestrating such an attack render it virtually infeasible. The required resources to compromise a meaningful number of nodes outweigh any potential short-term gains, especially considering the defensive measures embedded within the Bitcoin protocol.

(3) Manipulation of wtxid: The wtxid is derived from a double SHA-256 hash of the entire transaction data, including witness information introduced by SegWit. This cryptographic process ensures that any alteration to the transaction data, even minor changes, results in a completely different wtxid due to the properties of the SHA-256 hash function. Attempting to modify transaction content while preserving the wtxid would require finding a collision in the SHA-256 hash function and is computationally unfeasible with current technology. Achieving such a collision requires immense computational power and time. Therefore, this attack is practically impossible.

(4) SegWit Adoption and Support: A critical factor for the feasibility of our scheme is the prevalence of SegWit transactions. While historical metrics regarding the stock of UTXO might suggest lower adoption rates due to dormant legacy coins, the flow of newly generated transactions tells a different story.

According to recent network statistics, the adoption rate of SegWit for new transactions has surpassed 95% as of late 2025 (https://transactionfee.info/charts/payments-spending-segwit/). This ubiquity is further driven by the widespread adoption of Taproot (SegWit v1) and the emergence of witness-dependent protocols such as Ordinals. Consequently, our optimization covers the vast majority of active network traffic. For the remaining small fraction of legacy transactions, our system seamlessly reverts to the standard validation process, ensuring complete backward compatibility without compromising system stability.

(5) Network Resilience: Bitcoin's decentralized nature and the redundancy of having numerous independent nodes contribute significantly to network resilience. These characteristics ensure that the network can with-stand localized attacks or node failures without com-promising overall security and functionality. With the majority of nodes supporting SegWit and recognizing wtxid, the network exhibits a high degree of resilience against attempts to introduce inconsistent validation rules. The consensus mechanism ensures that almost all deviations from recognized protocols are resisted and corrected through the collective agreement of honest nodes. The proposed optimization aligns with the existing consensus rules and the widespread adoption of SegWit. As such, it leverages the network's inherent resilience and decentralization to implement efficiency improvements without undermining security.

In conclusion, the security analysis demonstrates that our proposed optimization maintains robust security assurances. By leveraging wtxid to simplify the second validation step, the scheme preserves the integrity of the current network framework. The decentralized and redundant nature of the Bitcoin network, coupled with the cryptographic strength of wtxid, significantly reduces the feasibility of successful attacks aimed at undermining transaction integrity. While the current analysis suggests minimal risk, it is prudent to implement additional safeguards such as enhanced monitoring of transaction pools and the incorporation of historical transaction data during validation. These measures can further mitigate residual risks associated with simplifying the second validation step.

5. Discussion

A. Comparative Analysis: Deterministic *vs.* Probabilistic Optimization

A core contribution of this paper is the shift from probabilistic caching to deterministic indexing. We explicitly compare our proposed scheme with the native Bitcoin Core caching mechanisms described in Section 2-E.

- Probabilistic Limitations of Existing Caches: As established in Section 2, the ValidationCache operates on a probabilistic basis. Its effectiveness relies on the cache hit rate, which fluctuates with memory availability and transaction volume. In "worst-case" scenarios (e.g., Mempool spam at-tacks or low-memory nodes), cache evasion occurs, forcing redundant computations.
- Deterministic Guarantee of Our Scheme: In contrast, our proposed scheme leverages the Mempool existence index (via "wtxid"). Since the Mempool acts as the mandatory staging area for unconfirmed transactions, the presence of a transaction in the Mempool serves as a deterministic proof of prior validation. As long as a transaction resides in the Mempool, our algorithm guarantees that redundant checks are skipped, independent of cache eviction policies or history. This provides a stable $O(1)$ validation path that is mathematically robust against cache thrashing.

B. Insights from Community Technical Discussions

To ensure the robustness of our theoretical framework, we consulted with the Bitcoin development community (https://groups.google.com/g/Bitcoindev/c/CkrS-L4vL7k/m/mPZC6JbeAQAJ) regarding the system's current validation logic. Core developers clarified the role of the ValidationCache and SignatureCache in mitigating redundant checks.

Based on this feedback and our subsequent analysis, we contend that while the existing caching infrastructure is highly effective for general-purpose optimization, it remains fundamentally probabilistic. We argue that our proposed Mempool-based indexing provides a necessary deterministic complement to the existing caches. By explicitly distinguishing between "cached validation" (which can be evicted) and "Mempool-resident status" (which is stateful and persistent until confirmation), our scheme offers a theoretically more stable optimization for the specific phase of block propagation.

C. Limitations and Real-World Challenges

While our experiments show significant efficiency gains, we must acknowledge that this study relies on a controlled simulation. The live Bitcoin network is more complex than a simulation. Factors such as network delays, bandwidth limits, and varying hardware capabilities are difficult to replicate perfectly. Specifically, in a real P2P network, memory pools do not sync instantly across all nodes. This delay can cause temporary differences between a miner's block and a verifier's local pool. As a result, the "hit rate" in the real world might be lower than in our simulation. Additionally, our current model does not fully cover potential attacks, such as malicious nodes trying to bypass the cache (DoS attacks).

Future work will focus on three key directions. First, we aim to further test the security and stability of the proposed algorithm in complex network environments to ensure it can withstand potential attacks. Second, in addition to the methods outlined in this paper, we will explore the verification of transaction legitimacy and the security of data transmission throughout its entire lifecycle. Third, we will deploy the prototype on the Bitcoin Testnet to evaluate its practical effectiveness and performance under real-world conditions.

6. Conclusion

Bitcoin, as the leading cryptocurrency by market capitalization, has maintained significant attention from both academic and industrial sectors over the past 16 years. Despite its widespread adoption and strong security features, Bitcoin faces performance challenges, particularly in transaction validation, which impede its scalability and efficiency. This study thoroughly examined the life cycle of Bitcoin transactions and the new block validation process by analyzing the Bitcoin source code. We identify redundant validation operations as a major bottleneck limiting the system's overall efficiency. To address this, we propose a novel optimization scheme leveraging SegWit technology to streamline the transaction validation process. Our approach introduces a one-time validation simplification mechanism that eliminates the need for the second validation of transactions already present in the local transaction pool when processing new blocks. Theoretical analysis and experimental results demonstrate that this optimization can cut down the transaction validation time by nearly half, thereby enhancing efficiency without compromising the security and integrity of the Bitcoin network.

The implications of our findings are substantial for the sustainability and scalability of the Bitcoin ecosystem. By mitigating inherent performance limitations, our optimization facilitates faster block propagation and more efficient transaction processing. This ensures Bitcoin is better equipped to handle increasing transaction volumes and evolving security demands. Future research will focus on empirical evaluations of the proposed optimization in real-world environments and exploring strategies that balance speed and network reliability. Collaborating with the Bitcoin community, we aim to

implement and validate these optimizations, contributing to the ongoing evolution and robustness of the Bitcoin protocol.

Data availability statement

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

Declaration of generative AI and AI-assisted technologies

During the preparation of this manuscript, the authors used generative AI tools only to improve language and readability. The authors take full responsibility for the content of the manuscript.

Acknowledgments

This work is supported by the Primary Research & Development Plan of Jiangsu Province (No. BE2023025-2) and the National Natural Science Foundation of China under Grants 62472209 and 92582106.

Authors' contribution

Conceptualization, Shunqing Xu and Changhai Nie; methodology, Shunqing Xu; software, Yanqi Xu and Dawei Nie; validation, Tianang Yao, Lili Yao and Huayao Wu; formal analysis, Shunqing Xu; investigation, Chen Tian; resources, Changhai Nie; data curation, Yanqi Xu; writing—original draft preparation, Shunqing Xu; writing—review and editing, Changhai Nie; visualization, Shunqing Xu; supervision, Changhai Nie; project administration, Changhai Nie; funding acquisition, Changhai Nie. All authors have read and agreed to the published version of the manuscript.

Conflicts of interest

The authors declare no conflicts of interest.

References

[1] Nakamoto S. Bitcoin: a peer-to-peer electronic cash system. 2008. Available: http://Bitcoin.org/Bitcoin.pdf (accessed on 13 March 2024).

[2] Lasla N, Al-Sahan L, Abdallah M, Younis M. Green-PoW: an energy-efficient blockchain proof-of-work consensus algorithm. *Comput. Networks* 2022, 214:109118.

[3] Yin H, Zhang Z, He J, Ma L, Zhu L, *et al*. Proof of continuous work for reliable data storage over permissionless blockchain. *IEEE Internet Things J.* 2021, 9(10):7866–7875.

[4] Bagaria V, Kannan S, Tse D, Fanti G, Viswanath P. Prism: deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, London, UK, November 11–15, 2019, pp. 585–602.

[5] Naumenko G, Maxwell G, Wuille P, Fedorova A, Beschastnikh I. Erlay: efficient transaction relay for Bitcoin. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, London, UK, November 11–15, 2019, pp. 817–831.

[6] Gao F, Mao H, Wu Z, Shen M, Zhu L, *et al*. Lightweight Bitcoin transaction traceability mechanism. *Chin. J. Comput.* 2018, 41(5):899–1004.

[7] Yang J, Wang H, Gao Z, Guo Z. Stateless transaction verification scheme based on dual RSA accumulators (In Chinese). *J. Zhejiang Univ.* 2023, 57(1):178–189.

[8] Erdin E, Cebe M, Akkaya K, Solak S, Bulut E, *et al*. A Bitcoin payment network with reduced transaction fees and confirmation times. *Comput. Networks* 2020, 172:107098.

[9] Dai X, Xiao B, Xiao J, Jin H. An efficient block validation mechanism for UTXO-based blockchains. *In Proceedings of 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Lyon, France, May 30–June 3, 2022, pp. 1250–1260.

[10] Kedziora M, Pieprzka D, Jozwiak I, Liu Y, Song H. Analysis of segregated witness implementation for increasing efficiency and security of the Bitcoin cryptocurrency. *J. Inf. Telecommun.* 2023, 7(1):44–55.

[11] Chin ZH, Baskaran VM, Tan CK, Tan IKT, Yap TTV. DAP-CBR: enhancing Bitcoin block propagation efficiency using dynamic compact block relay's prefilling of transactions. *J. Supercomput.* 2025, 81(1):77.

[12] Anjana PS, Amini M, Kapoor R, Parmar R, Ramesh R, *et al*. Efficient parallel execution of blockchain transactions leveraging conflict specifications. In *Proceedings of 7th Conference on Advances in Financial Technologies (AFT 2025)*, Pittsburgh, USA, October 8–10, 2025, pp. 1–29.

[13] Wen J, Kang J, Xiong Z, Du H, Yang Z, *et al*. Optimal AoI-based block propagation and incentive mechanism for blockchain networks in Web 3.0. *IEEE Trans. Cognit. Commun. Networking* 2024, 11(4):2568–2583.

[14] Reddy BS, Reddy TUK. CompactChain: an efficient stateless chain for UTXO-model blockchain. *Front. Comput. Sci.* 2024, 18(2):182806.

[15] Lombrozo E, Lau J, Wuille P. BIP141: Segregated Witness (Consensus Layer). 2015. Available: http://github.com/Bitcoin/bips/blob/master/bip-0141.mediawiki (accessed on 13 March 2024).

[16] Lau J. BIP142: Address format for Segregated Witness. Available: https://github.com/Bitcoin/bips/blob/master/bip-0142.mediawiki (accessed on 13 March 2024).

[17] Lau J, Wuille P. BIP143: Transaction signature verifi-cation for version 0 witness program. Available: https://github.com/Bitcoin/bips/blob/master/bip-0143.mediawiki (accessed on 13 March 2024).

[18] Lombrozo E, Wuille P. BIP144: Segregated Witness (Peer Services). Available: https://github.com/Bitcoin/bips/blob/master/bip-0144.mediawiki (accessed on 13 March 2024).

[19] Luke Dashjr. BIP145: getblocktemplate Updates for Segregated Witness. Available: https://github.com/Bitcoin/bips/blob/master/bip-0145.mediawiki (accessed on 13 March 2024).

[20] Andrychowicz M, Dziembowsik S, Malinowski D, Mazurek Ł. How to deal with mealleability of Bitcoin transactions. Available: https://arxiv.org/pdf/1312.3230.pdf (accessed on 13 March 2024).

[21] Nie C, Lu C, Gao W, Zhong Z. *Blockchain Technology Fundamentals: Principles, Methods, and Practices (In Chinese)*. Beijing: Machinery Industry Press, 2023.

[22] Antonopoulos AM, Harding DA. *Mastering Bitcoin: programming the open blockchain*, 3rd ed. Sebastopol: O'Reilly Media, Inc., 2023.

[23] Bashir I. *Mastering Blockchain: Inner workings of blockchain, from cryptography and decentralized identities, to DeFi, NFTs and Web3*, 4th ed. Birmingham: Packt Publishing Ltd., 2023.