electronics

MDPI

Article

DCSCY: DRL-Based Cross-Shard Smart Contract Yanking in a Blockchain Sharding Framework

Ying Wang [1], Zixu Zhang [2], Hongbo Yin [3], Guangsheng Yu [2], Xu Wang [2], Caijun Sun [4], Wei Ni [5], Ren Ping Liu [2] and Zhiqun Cheng [1,*]

1 School of Electronics and Information, Hangzhou Dianzi University, Hangzhou 310018, China; 90023@hdu.edu.cn
2 Global Big Data Technologies Centre, University of Technology Sydney, Ultimo 2007, Australia; zixu.zhang@alumni.uts.edu.au (Z.Z.); guangsheng.yu@uts.edu.au (G.Y.); xu.wang@uts.edu.au (X.W.); renping.liu@uts.edu.au (R.P.L.)
3 School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China; hongboyin@std.uestc.edu.cn
4 Zhejiang Lab, Hangzhou 311121, China; sun.cj@zhejianglab.com
5 Data61, CSIRO, Sydney 2015, Australia; wei.ni@data61.csiro.au
* Correspondence: zhiqun@hdu.edu.cn

Abstract

Blockchain sharding has emerged as a promising solution to address scalability and performance challenges in distributed ledger systems. In the sharded blockchain, yanking can reduce the communication overhead of smart contracts between shards. However, the existing smart contract yanking methods are inefficient, increasing the latency and reducing the throughput. In this paper, we propose a novel DRL-Based Cross-Shard Smart Contract Yanking (DCSCY) framework which intelligently balances three critical factors: the number of smart contracts processed, node waiting time, and yanking costs. The proposed framework dynamically optimizes the relocation trajectory of smart contracts across shards. This reduces the communication overhead and enables adaptive, function-level migrations to enhance the execution efficiency. The experimental results demonstrate that the proposed approach reduces the cross-shard transaction latency and enhances smart contract utilization. Compared to random-based and order-based methods, the DCSCY approach achieves a performance improvement of more than 95%.

Keywords: blockchain; sharding; smart contract; yanking; deep reinforcement learning

check for updates

Academic Editor: Fabio Grandi

Received: 9 July 2025
Revised: 11 August 2025
Accepted: 15 August 2025
Published: 16 August 2025

Citation: Wang, Y.; Zhang, Z.; Yin, H.; Yu, G.; Wang, X.; Sun, C.; Ni, W.; Liu, R.P.; Cheng, Z. DCSCY: DRL-Based Cross-Shard Smart Contract Yanking in a Blockchain Sharding Framework. *Electronics* **2025**, *14*, 3254. https://doi.org/10.3390/electronics14163254

1. Introduction

Blockchain technology [1–5] has emerged as a transformative paradigm that enables decentralized, transparent, and tamper-resistant platforms for digital transactions and smart contract execution. Among the most prominent blockchain systems, Bitcoin [6] pioneered decentralized digital currency, and Ethereum [7] advanced the field by introducing programmable smart contracts. These innovations have paved the way for a wide range of applications, including decentralized applications (dApps) [8], decentralized finance (DeFi) [9,10], supply chain management [11], and Internet of Things (IoT) integration [12,13]. Despite these advancements, scalability and performance remain major bottlenecks for traditional blockchain systems. As blockchain adoption continues to grow and transaction volumes surge, enhancing the system throughput and execution efficiency has become a critical imperative.

Sharding has emerged as a promising solution to address blockchain scalability issues [14–16]. By partitioning the blockchain network into smaller, independently processing subsets (shards), the system can achieve a high throughput via parallel transaction execution. However, sharding introduces the complexity of cross-shard communication, where transactions or smart contracts need to interact across different shards. The conventional solutions employ locking mechanisms to maintain consistency, but these approaches incur a high communication overhead and synchronization delays [17–19].

To address these challenges, the concept of yanking [20,21] has emerged as an alternative. Yanking dynamically relocates smart contracts between shards to enable real-time function-level relocation. This approach optimizes the transaction flow and reduces the waiting times associated with inter-shard dependencies. As demonstrated in Table 1, yanking mechanisms enhance system efficiency by avoiding the multi-block delays associated with locking mechanisms. Despite its efficiency advantages, yanking is not suitable for consortium blockchains because its dynamic relocation of smart contracts across shards compromises shard-level data confidentiality [22]. Additionally, yanking mechanisms do not optimize the resource allocation, as they relocate smart contracts without considering shard-specific demands. This can lead to prolonged waiting times for other shards requiring access to the same contract.

Table 1. Comparison of locking and yanking mechanisms in sharded blockchain smart contract execution.

Feature	Locking [17–19]	Yanking [20,21]
Mechanism	Holds contract in one shard, preventing access by others	Immediate-execution contract relocation to target shard
FunctionScope	Contract-level	Function-level
Latency	Higher, due to synchronization delays across shards	Lower, as it avoids long multi-block synchronization
Overhead	Higher due to cross-shard locks	Reduces inter-shard communication complexity
Consistency	Maintains consistency with locks	Ensures consistency through real-time relocation
IdealUse	Critical-consistency low-frequency cross-shard interactions	High-frequency rapid cross-shard processing

To address these limitations, we propose an innovative approach leveraging deep reinforcement learning (DRL) to optimize the trajectory of smart contract yanking, enabling efficient and adaptive relocations across shards. The proposed DRL-Based Cross-Shard Smart Contract Yanking (DCSCY) framework maximizes the overall system efficiency by intelligently balancing three critical factors: the number of processed smart contracts, node waiting times, and yanking costs. By dynamically predicting and selecting the optimal yanking trajectory, this approach effectively addresses the inefficiencies present in existing yanking mechanisms. This work makes the following key contributions:

1. DCSCY is proposed, making this the first study to apply DRL techniques to optimizing the yanking trajectory of smart contracts in sharded blockchain frameworks. By leveraging DRL, DCSCY dynamically learns and predicts the optimal sequence for contract relocations, improving the execution efficiency.
2. The DCSCY framework effectively balances three critical factors: the number of processed smart contracts, the waiting time for nodes requiring contract execution, and the communication cost associated with yanking.

3. The experimental results show that DCSCY improves the performance by more than 95% compared to that of order-based and random-based yanking methods. These results highlight the effectiveness of DCSCY in enhancing smart contract execution efficiency and reducing system congestion in sharded blockchain environments.

The remainder of this paper is organized as follows: Section 2 reviews related work on cross-shard smart contract transactions, locking versus yanking mechanisms, and the application of DRL in sharded blockchains. Section 3 introduces the proposed system model and yanking mechanism, while Section 4 details the DRL framework used to optimize smart contract relocation. Section 5 presents the experimental results and performance evaluations. Finally, Section 6 concludes this paper and discusses potential future research directions.

2. Related Work

In this section, we provide an overview of the existing approaches to addressing the challenges of cross-shard communication in sharded blockchain systems, with a particular focus on smart contract execution. Sharding has proven effective in enhancing blockchain scalability, but the latency and overhead associated with cross-shard transactions remain significant bottlenecks. We analyze state-of-the-art solutions in three key areas: cross-shard smart contract transactions, the transition from smart contract locking to yanking, and the application of DRL to optimizing shard management and contract relocation. This review lays the foundation for understanding how DCSCY advances the current methodologies.

2.1. Cross-Shard Smart Contract Transactions in the Sharded Blockchain

Smart contracts are distributed across shards to balance the computational load, with locking mechanisms used to maintain consistency during cross-shard interactions. Protocols such as Atomic [23] and S-BAC [24] employ these locking mechanisms to coordinate cross-shard transactions and ensure data integrity. Although locking mechanisms effectively ensure data integrity, they can also result in high cross-shard transaction (CSTx) ratios due to frequent interactions between accounts and contracts across different shards. In addition, uncertainty in the transaction processing times complicates execution flows and increases the communication overhead. As a result, multiple execution steps across shards are required, which leads to significant latency and decreased system efficiency.

To address the challenges of CSTxs in sharded blockchain systems, researchers have proposed various strategies. Zhang et al. [25] propose an overlapping shard architecture that converts CSTxs into intra-shard transactions by leveraging overlapping shard nodes to simplify transaction management and reduce the cross-shard communication latency. Li et al. [26] propose the Jenga system, which shares the execution logic of all smart contracts across shards, allowing overlapped nodes to broadcast contract states directly between state shards and execution channels. Jenga eliminates the need for additional cross-shard communication, thereby enhancing the throughput and reducing the transaction confirmation latency. Qi et al. [27] present LightCross, which periodically migrates frequently interacting smart contracts to the same shard, minimizing the CSTx ratio and using a lightweight cross-shard commit protocol for efficient handling of residual CSTxs. Additionally, Liang et al. [28] introduce SPARROW, a sharding protocol that accelerates cross-shard smart contract execution through an efficient one-step execution mechanism supported by inter-shard caching, speculative cache synchronization, and multi-branch exploration for rollback handling. These approaches collectively demonstrate the potential of overlapping architectures, dynamic contract migration, and inter-shard caching for optimizing cross-shard smart contract execution.

2.2. Smart Contracts from Locking to Yanking

Traditional cross-shard transactions in the sharded blockchain rely on locking mechanisms to maintain the execution consistency across multiple shards. Locking protocols provide strong guarantees of atomicity and consistency by preventing conflicting operations on smart contracts. Dang et al. [17] propose a two-phase locking (2PL) protocol to ensure atomicity and isolation in cross-shard transactions. When applied to sharded blockchain systems, 2PL ensures that cross-shard transactions are executed atomically by locking all involved contracts until the transaction completes. However, this method can lead to long transaction wait times and reduced throughput, particularly when malicious coordinators induce infinite blocking scenarios. To overcome the limitations of traditional locking mechanisms, Huang et al. [18] introduce the fine-tuned lock protocol, which addresses these inefficiencies by allowing for partial real-time transaction processing during account migrations. Unlike conventional locks that block all related transactions, the fine-tuned lock only locks the payer transactions, allowing payee transactions to proceed in real time during the account migration process. Although these locking mechanisms maintain smart contract integrity, they often fall short in terms of their scalability and efficiency, especially under high cross-shard smart contract transaction throughput conditions.

To overcome these limitations, Wels [7,29] propose smart contract yanking as an alternative mechanism to ensure transaction atomicity for Ethereum 2.0. The yanking process in Ethereum 2.0 is initiated when a transaction requests the relocation of a smart contract to the shard that requires its execution [30]. Once approved, the source shard generates a receipt containing the state of the contract and the identifier of the requesting shard. The receipt, along with a Merkle proof, is transmitted to the requesting shard, which imports the contract state and completes the relocation. This approach achieves execution consistency by dynamically relocating contracts to the shards where execution is required, thereby eliminating the need for prolonged locking and reducing the cross-shard communication overhead. Zamani et al. [31] propose RapidChain, which uses a yanking mechanism but focuses on relocating UTXOs (Unspent Transaction Outputs) rather than smart contracts. In RapidChain, the input transactions are first moved from the input shard to the output shard. Once all inputs are transferred, the final transaction is executed on the output shard.

2.3. The DRL-Based Sharded Blockchain

The application of DRL in blockchain research has garnered increasing attention due to its ability to optimize the decision-making processes in complex, decentralized environments. For instance, Yu et al. [32] propose a novel multi-agent DRL framework for resource scheduling in permissionless sharded blockchains. Their Rainbow-WoLF-PHC algorithm integrates a Rainbow Deep Q-Network (DQN) and WoLF-PHC, achieving rapid convergence with mixed-strategy Nash equilibrium. This approach optimizes the resource allocation but does not address smart contract interactions or the relocation efficiency. Yang et al. [33] introduce DRL-OSS, an overlapping, self-organizing sharding scheme optimized through a single-agent DRL model. This scheme maximizes the throughput and security by dynamically adjusting sharding strategies, thereby reducing CSTs. However, it does not consider the dynamic relocation of smart contracts, leading to potential inefficiencies when executing interdependent contracts across shards. Li et al. [34] propose SPRING, a DRL-based state placement framework that models the state placement as a Markov Decision Process (MDP). By leveraging the spatiotemporal characteristics of transactions, SPRING effectively reduces the CST ratio and enhances the throughput while maintaining workload balance. However, SPRING focuses on static state placement and does not dynamically relocate smart contracts based on the interaction frequency.

Zhang et al. [15] introduce TbDd, a trust-driven, DRL-based framework that optimizes sharding by integrating trust evaluation with DRL algorithms like a DQN [35] and Proximal Policy Optimization (PPO) [36]. While TbDd reduces CSTs, balances the node distribution, and enhances the throughput, it focuses on node allocation and security without optimizing the smart contract execution efficiency.

Building on these developments, the DCSCY system utilizes DRL to optimize the relocation of smart contracts between shards, minimizing the latency and computational overhead associated with cross-shard communication. Unlike existing DRL-based sharding strategies [15,32–34] that focus on resource allocation and state placement, DCSCY addresses the dynamic relocation of smart contracts required for efficient yanking mechanisms. The existing yanking mechanisms designed in Ethereum 2.0 [20,21] incur high communication costs and latency due to frequent contract relocations. To address this, DCSCY utilizes DRL's adaptive learning capability to predict the optimal shard placements by analyzing historical transaction patterns, thereby reducing cross-shard transaction ratios and the communication overhead.

3. The System Model: Yanking Smart Contract Calling Based on DRL

This section presents the architecture of the proposed smart contract calling methods with a yanking mechanism, designed to support a sharded blockchain framework. It defines the roles involved in the system, outlines the workflow, and clarifies the underlying system assumptions.

3.1. The System Overview

The System Model. Figure 1 presents a system model of the proposed DCSCY framework within a sharded blockchain environment. The system consists of multiple shards, such as S_1, S_2, and S_3, where each S_i denotes an independent chain composed of all nodes participating in the i-th shard. To support efficient cross-shard execution, DCSCY introduces a Yanking Community (YC), composed of selected shard leaders based on reliability and performance metrics. Each leader deposits a collateral fee to deter malicious behavior and ensure accountability. Acting as the DRL agent, the YC maintains a global view of the smart contract invocation requests and node queuing times across all shards, enabling informed and collaborative yanking decisions. In each training episode, denoted as ε, the system predicts the trajectory of contract relocations based on the current network conditions. An episode includes γ blocks. If the agent decides to relocate a contract α times, the episode results in $\alpha\gamma$ blocks being proposed across the shards. This dynamic contract migration allows the system to reduce the synchronization overhead and improve the execution latency.

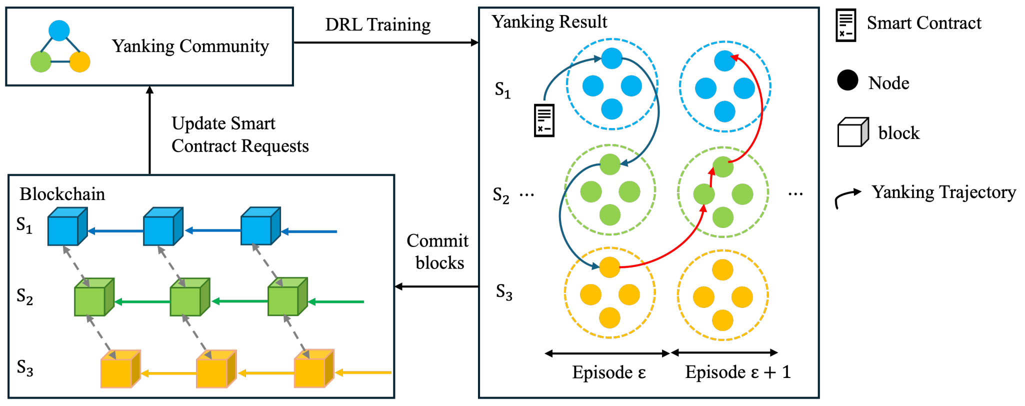

Figure 1. A system model of the proposed cross-shard smart contract yanking—DCSCY.

Based on the YC-coordinated approach, the DCSCY framework adopts a dynamic yanking mechanism to relocate smart contracts between shards. Unlike locking mechanisms, the yanking mechanism prioritizes function-level operations by temporarily transferring the contract state to the target shard for execution. Once execution is complete, the state is updated and securely returned to the originating shard, accompanied by a receipt verifying the correctness of the state transition. This verification process is reinforced with a Merkle proof from the account user to ensure the integrity of the transition.

Architecture. Figure 2 illustrates the architecture of the DCSCY framework. The black arrows between blocks indicate that each block contains the hash of its preceding block. In the left section, the upper part represents the historical trajectory of smart contract yanking during episode ε, with the yanking process represented by blue arrows. In the right section, the YC employs the DRL algorithm to predict the optimal yanking trajectory, represented by the red arrows, for the next episode $\varepsilon + 1$. At the beginning of the environment, the smart contract is yanked to the shard with the highest number of requests for a smart contract. If multiple shards have an equal number of requests, the contract is randomly yanked to one of them.

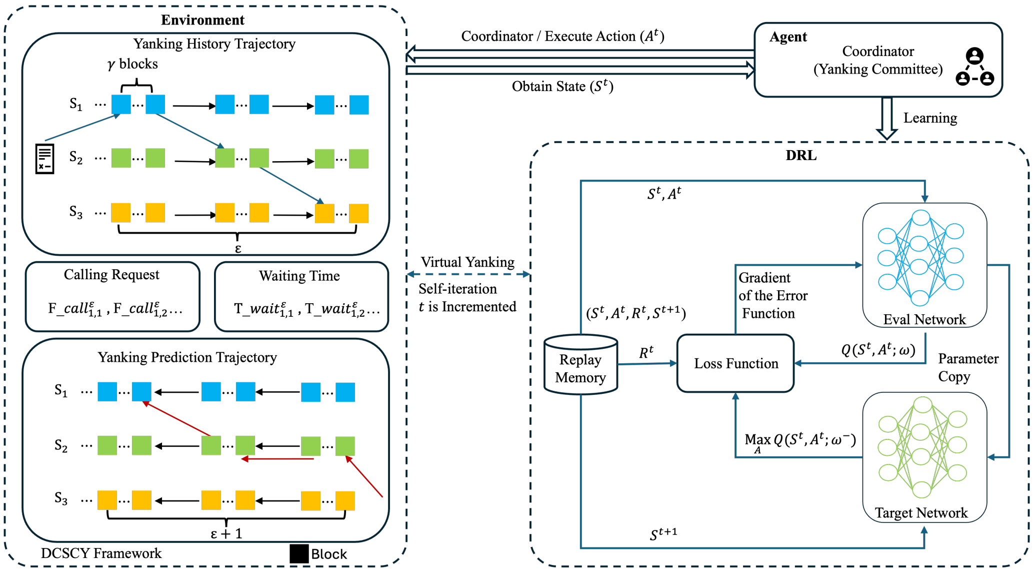
Environment
Yanking History Trajectory
γ blocks
S1
S2
S3
ε
Calling Request
F_call$^{\varepsilon}_{1,1}$, F_call$^{\varepsilon}_{1,2}$...
Waiting Time
T_wait$^{\varepsilon}_{1,1}$, T_wait$^{\varepsilon}_{1,2}$...
Yanking Prediction Trajectory
ε + 1
DCSCY Framework
Block
Coordinator / Execute Action (A^t)
Obtain State (S^t)
Virtual Yanking
Self-iteration
t is Incremented
Agent
Coordinator
(Yanking Committee)
Learning
DRL
S^t, A^t
(S^t, A^t, R^t, S^{t+1})
Gradient of the Error Function
Eval Network
Replay Memory
R^t
Loss Function
$Q(S^t, A^t; \omega)$
Parameter Copy
$\mathrm{Max}_A\, Q(S^t, A^t; \omega^-)$
Target Network
S^{t+1}

Figure 2. The architecture of the DCSCY framework.

During each episode, the YC collects data on the calling requests and waiting times. Calling requests refer to the number of transactions initiated by the n_k nodes within the k-th shard that require the smart contract during episode ε, denoted as $F\_call^{\varepsilon}_{k,n\_k}$. The waiting time represents the total delay experienced by n_k nodes in the k-th shard while waiting for the smart contract to be yanked during episode ε, denoted as $T\_wait^{\varepsilon}_{k,n\_k}$, and is calculated in (2). Based on the learned policy, the smart contract is dynamically migrated to the shard that minimizes the overall request congestion and waiting delays, thereby improving the system's responsiveness and throughput.

Roles. In the DCSCY framework, the network consists of N nodes organized into K shards, where each node is responsible for proposing blocks to its peers within the same shard for validation. Additionally, nodes function as validators, actively participating in the validation process within the sharded blockchain. Notably, the framework minimizes the leadership role of the nodes, as DCSCY emphasizes decentralized and efficient smart contract execution verification and optimization of the smart contract yanking across shards, rather than centering on the consensus mechanism.

3.2. Workflow Overview

The proposed DCSCY framework follows the workflow in Figure 3.

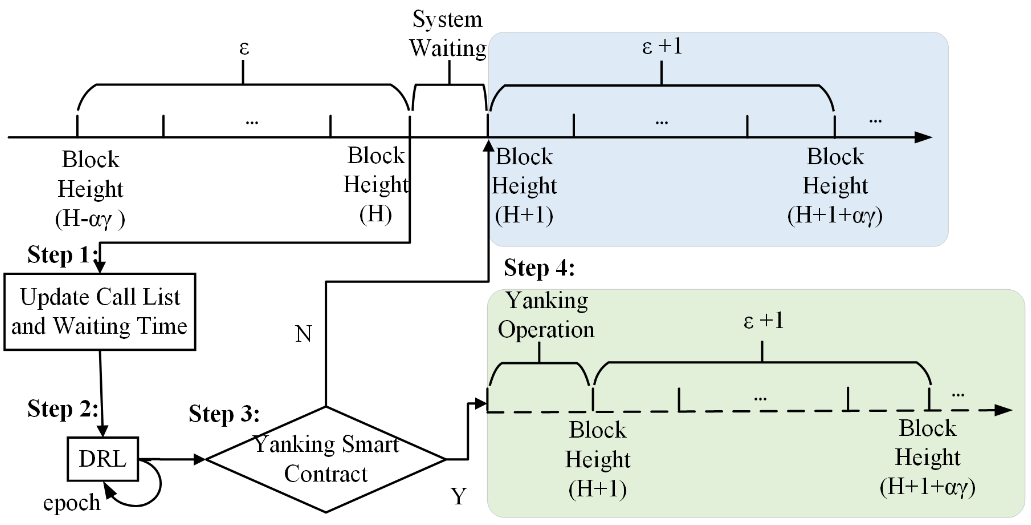

Figure 3. The proposed blockchain sharding system flowchart.

Step-1. Update the calling list and waiting time. The first step in the framework is initializing with the smart contract deployed to a shard based on a calling request. The smart contract calling request and waiting times are updated for all shards in each episode (every $\alpha\gamma$ blocks), where the calling list and the waiting time for each shard are refreshed based on the latest transaction data collected during this interval.

Step-2. Train the model with the DRL algorithm. Acting as the agent, the YC trains the model using the collected data to evaluate the current state. This involves utilizing the DRL-based model to simulate smart contract yanking decisions and output the optimal shard placement for the smart contract.

Step-3. Determine whether to yank the smart contract. The YC iteratively calculates rewards over multiple epochs to determine the optimal action. Based on the training results and DRL model predictions, it calculates the rewards for each action and decides whether to yank the smart contract to a new shard.

Step-4. The yanking operation. The smart contract is yanked to the new shard, as determined by the DRL agent. This executes the yanking operation to relocate the smart contract to the optimal shard and completes related transactions.

During this period, as the YC agent retrains the DRL model to re-evaluate the optimal shard placement for the smart contract, the system remains in an offline state. Once training is complete, the system resumes operation, and the cycle restarts from Step-1, progressing through to Step-4 iteratively.

3.3. System Assumptions

The proposed DCSCY system is designed with the following assumptions:

Protocol Adherence by All Users: DCSCY assumes that all users strictly adhere to the established protocol rules to ensure smooth execution of the yanking operations and to minimize disruptions caused by user misbehavior.

Uniformity and Universal Applicability: The system assumes that all nodes are homogeneous, possessing equal computational power and communication resources, which eliminates disparities and ensures seamless operation. Moreover, DCSCY is universally applicable to various blockchain types, including public blockchains, private blockchains, and consortium blockchains.

Committee-Driven Governance and Leader Accountability: Leaders are chosen based on a transparent selection process, considering factors such as node reliability and predefined rotation policies to ensure fairness and prevent centralization. The execution of yanking operations is managed by the YC composed of selected leaders, who employ DRL algorithms to optimize the decision-making and ensure efficient system operation. To maintain security and trust within the network, misbehaving leaders face strict penalties, including monetary fines or removal from the network.

4. The DRL Framework

The proposed DRL framework is designed to optimize the trajectory of smart contract yanking in a sharded blockchain system, ensuring efficient cross-shard deployment while minimizing the latency and communication overhead, as shown in Algorithm 1. By leveraging the dynamic learning capabilities of DRL, the framework continuously adapts to the network conditions, transaction demands, and shard-specific states, making real-time decisions for optimal contract relocation. The objective is to maximize the overall system efficiency while balancing the execution speed, waiting times, and yanking costs.

Algorithm 1: DQN-based yanking algorithm

```
Input:
  $N$: Total number of nodes;
  $K$: Total number of shards;
  $F\_call_{k,n\_k}$: Calling request number for the $n\_k$-th node in the $k$-th shard;
  $T\_wait_{k,n\_k}$: Waiting time for the $n\_k$-th node in the $k$-th shard;
  $c\_yank$: Yanking cost when yanking the contract between shards;
  $\lambda_1, \lambda_2, \lambda_3$: Weights of $S\_exe_k$, $S\_wait_k$, and $C\_yank_k$;
  $\xi$: Discount factor;
  $\tau$: Target network update rate;
1  Initialize:   Policy network $Q_\omega$, target network $Q_{\omega^-}$;
2    Replay buffer $\mathcal{M}$;
3    State vector $s_t = [\text{F\_call}^t_{k,n\_k}, \text{T\_wait}^t_{k,n\_k}, y_h]$;
4  for each epoch do
5    Observe initial state $s_0$;
6    for each step $t$ do
7      Select action $a_t$ via $\epsilon$-greedy;
8      Execute $a_t$, compute reward: $r : R(S\_exe, S\_wait, C\_yank)$;
9      Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{M}$;
10     if $\mathcal{M}$ is full then
11       Sample mini-batch $\mathcal{B} \sim \mathcal{M}$;
12       Compute target Q-values: $q_i = r_i + \xi \max_{a'} Q_{\omega^-}(s_{i+1}, a')$;
13       Update $Q_\omega$ via gradient descent: $\omega \leftarrow \omega - \beta \nabla_\omega \frac{1}{|\mathcal{B}|} \sum (Q_\omega(s,a) - q)^2$;
14       Soft-update the target network: $\omega^- \leftarrow \tau\omega + (1-\tau)\omega^-$;
15     Update state;
16   Decay exploration rate $\epsilon \leftarrow \epsilon \cdot e^{-d}$;
17 return $y'_{h+1} \cdots y'_{h+\alpha}$.
```

4.1. Optimizations, Rewards, and DRL

In the DCSCY framework, DRL is employed to optimize the trajectory of smart contract yanking. The primary objective of integrating DRL is to dynamically predict the optimal shard for repositioning smart contracts within each episode. By leveraging historical data and real-time state information, the DRL model intelligently learns and adapts to varying network conditions. Specifically, it predicts the smart contract's position in the current episode based on the state of the shard, including smart contract calling requests across different shards, the node waiting times in each shard, and the associated yanking costs. The reward function is designed to evaluate and prioritize actions that enhance the execution efficiency of smart contract transactions while simultaneously minimizing the latency incurred by cross-shard interactions. Therefore, the DRL-based model plays a critical role in determining the optimal shard for smart contract yanking during each episode.

In this section, we propose a single agent of the DCSCY framework. The agent aims to maximize the earnings by calculating the objective function that is composed of 3 reward components, where S_exe, S_wait and C_yank are all constant. The notation is shown in Table 2.

Table 2. Notation and definition used in the DCSCY framework.

Notation	Description
N	Number of nodes in the network
K	Number of shards in the network
ε	Fixed episode for state update
$F\_call^{\varepsilon}_{k,n\_k}$	Smart contract calling requests by n_k-th node in k-th shard during the ε episode
$S\_call^{\varepsilon}_{k}$	Total smart contract calling requests by k-th shard during ε episode
$T\_wait^{\varepsilon}_{k,n\_k}$	Waiting time for n_k-th node in k-th shard to call smart contract during ε episode
$S\_wait^{\varepsilon}_{k}$	Cumulative waiting time of k-th shard to call smart contract during ε episode
y_h	Historical trajectory of smart contract yanking
$y'_{h+\alpha}$	Predicted trajectory of smart contract yanking
C_yank	Yanking cost, including the yanking operation (adding, deleting, and communication)
c_yank	Constant value for each yanking operation

The Number of Smart Contract Transactions Executed (S_exe): The number of smart contracts executed in the k-th shard during the ε episode is computed as follows:

$$S\_exe^{\varepsilon}_{k} = \begin{cases} (S\_exe^{\varepsilon}_{k})_{max}, & \text{if} \quad S\_call^{\varepsilon}_{k} \geq (S\_exe^{\varepsilon}_{k})_{max} \\ S\_call^{\varepsilon}_{k}, & \text{if} \quad S\_call^{\varepsilon}_{k} < (S\_exe^{\varepsilon}_{k})_{max} \end{cases} \tag{1}$$

where $S\_call^{\varepsilon}_{k} = \sum_{n\_k=1}^{n\_k} F\_call^{\varepsilon}_{k,n\_k}$ denotes the total number of transactions in the k-th shard calling the smart contract during the ε-th episode. $(S\_exe^{\varepsilon}_{k})_{max}$ represents the maximum number of smart contract transactions executed by the k-th shard during the ε-th episode.

The Waiting Time of the Shard (S_wait): The waiting time reflects the cumulative delay experienced by the nodes in a shard while waiting for the smart contract to be yanked. It is calculated as

$$T\_wait^{\varepsilon}_{k,n\_k} = \begin{cases} T\_wait^{\varepsilon-1}_{k,n\_k} + \sum_{n\_k=1}^{k} F\_call^{\varepsilon}_{k,n\_k} \times E\_wait^{\varepsilon}_{k,n\_k}, & \text{if no yanking} \\ 0, & \text{if yanking} \end{cases} \tag{2}$$

where $T\_wait^{\varepsilon}_{k,n\_k}$ represents the waiting time for the n_k-th node in the k-th shard to call the smart contract, and $E\_wait^{k}_{n\_k}$ denotes the waiting episodes for the n_k-th node in the k-th shard during the ε episode. As the waiting episodes increase, $T\_wait^{\varepsilon}_{k,n\_k}$ grows exponentially. When the smart contract is yanked to a shard, $T\_wait^{\varepsilon}_{k,n_k}$ resets to 0.

$$S\_wait^{\varepsilon}_{k} = \sum_{n\_k=1}^{n\_K} T\_wait^{\varepsilon}_{k,n\_k} \tag{3}$$

where $S\_wait^{\varepsilon}_{k}$ denotes the total waiting time of the k-th shard to call the smart contract.

The Yanking Cost per Yank (C_yank): This metric quantifies the overhead incurred each time a smart contract is relocated between shards.

$$C\_yank^{\varepsilon} = \begin{cases} 0, & \text{if } y_h = y_{h-1} \\ c\_yank, & \text{else } y_h \neq y_{h-1} \end{cases} \tag{4}$$

where C^{ε}_{yank} is the yanking cost, including the costs of adding contracts, deleting contracts, and communicating. In this paper, we assume that the yanking cost for each operation is a constant value c_yank.

Objective Function: Recall that the objective of the paper is to maximize the efficiency of the smart contract by yanking while minimizing delays. Thus, the objective function is defined as a weighted sum of various factors:

$$R = \sum_{j=1} (\lambda_1 S\_exe^{\varepsilon}_{k} + \lambda_2 S\_wait^{\varepsilon}_{k} - \lambda_3 C\_yank^{\varepsilon}_{k}) \tag{5}$$

$$\begin{aligned} &\text{Let } \mathbf{R} = [S\_exe^{\varepsilon}_{k}, S\_wait^{\varepsilon}_{k}, C\_yank^{\varepsilon}_{k}], \\ &\text{objective: } \max_{\mathbf{R}} \sum_{\varepsilon}^{\varepsilon_{max}} R(\mathbf{R}), \\ &\text{s.t.} \quad \lambda_1 + \lambda_2 + \lambda_3 = 1, \\ &\qquad\quad S\_wait^{\varepsilon}_{k} \leq \rho \end{aligned} \tag{6}$$

where the hyperparameters λ_1, λ_2, and λ_3 represent the weights of $S\_exe^{\varepsilon}_{k}$, $S\_wait^{\varepsilon}_{k}$, and C_yank^{ε}, respectively. ρ is a hyperparameter that limits the upper limit of the shard waiting time.

4.2. The DRL-Based Sharding Optimization Model

The YC is a committee composed of leaders selected from each shard. These leaders are responsible for executing the DRL training process. The YC collects data from all shards, including the number of times the smart contract is called, the waiting times for transactions involving the smart contract, and the yanking cost associated with relocating the contract to another shard. Using this data, the YC performs virtual yanking of the smart contract to simulate and evaluate the potential rewards. The training process enables the agent to determine the optimal shard for relocating the smart contract to maximize efficiency and minimize latency.

Agent: The agent, represented by the YC, collaboratively processes real-time data using the DRL model. By leveraging this data, the agent learns and predicts the optimal shard for the smart contract based on the historical yanking trajectories, smart-contract-related transaction demands, and yanking costs.

Environment: The environment, as illustrated in Figure 3, encompasses the sharded blockchain network and yanking operations. It provides the historical trajectory of the smart contract yanking process, which serves as the basis for evaluating the current state. Each new state is derived from the updated smart contract calling requests, node waiting times, and historical trajectory of the smart contract after a virtual yanking action. The environment feeds this information back to the agent to support the next decision-making cycle.

State ($\mathcal{S}$): The state $\mathcal{S}$ of each shard consists of three components. As shown in Figure 3, an evaluation of whether the smart contract will be yanked is performed after processing every $\alpha\gamma$ blocks. Factors like the node requests for the smart contract across shards, the waiting time, and the smart contract's yanking trajectory are considered. Overall, the state $\mathcal{S}$ is a $2Kn\_K + 1$-dimensional vector represented as follows:

$$\begin{aligned}\mathcal{S} =&[\mathrm{F\_call}^{\varepsilon}_{1,1}, \cdots, \mathrm{F\_call}^{\varepsilon}_{k,n\_k} \cdots, \mathrm{F\_call}^{\varepsilon}_{K,n\_K}, \\ &\mathrm{T\_wait}^{\varepsilon}_{1,1}, \cdots, \mathrm{T\_wait}^{\varepsilon}_{k,n\_k}, \cdots, \mathrm{T\_wait}^{\varepsilon}_{K,n\_K}, \\ &y_1, \cdots, y_h]\end{aligned} \tag{7}$$

where y_h is the historical trajectory of the smart contract.

Action ($\mathcal{A}$): Given a state $\mathcal{S}$, the agent selects an action $\mathcal{A}$. Specifically, the action $\mathcal{A}$ is an one-hot vector, represented as follows:

$$\mathcal{A} = [y'_{h+1}, \cdots, y'_{h+\alpha}] \tag{8}$$

where the value of y'_{h+1} is the postion of the smart contract in the next episode, and $y'_{h+\alpha}$ indicates the position of the smart contract in the next α episode.

Policy (π). A policy determines what action the agent will take next based on a set of rules. In this case, the process of yanking the smart contract to different shards is based on a policy that is continually updated and trained, i.e., $\pi(s, a) : \mathcal{S} \to \mathcal{A}$.

Reward Function (r). The reward function is inherited from the objective function, $r : R(S\_exe, S\_wait, C\_yank)$.

DRL plays a pivotal role in the DCSCY system by predicting the optimal shard placement for smart contract yanking. This dynamic prediction capability enables the system to minimize the latency and communication overhead, ultimately enhancing the overall efficiency of smart contract utilization. Unlike traditional methods such as convex optimization, DRL can effectively address the complex and dynamic nature of sharding. Its ability to adaptively learn from historical data and current states allows DRL to outperform the conventional approaches, offering more robust and efficient solutions to the challenges of smart contract yanking in sharded blockchain systems.

5. Experiments and Evaluation

This section presents the experimental setup and the design of the baseline methods used to evaluate the performance of the proposed DCSCY framework. The evaluation focuses on the convergence performance and execution stability under various configurations, including different numbers of shards and participating nodes. Based on the analysis of real-world Ethereum data [37], a single block on the Ethereum network typically contains between 100 and 200 transactions. Considering a period of 5 blocks as one episode, the total number of transactions in one episode is approximately 500–1000. As the scope of the analysis is limited to the yanking behavior of a single smart contract function, the number of transactions involving that specific contract is estimated to be up to 50 per episode. Assuming that these transactions involve calls to 10–20 different smart contract functions and as this study focuses on the yanking of a single smart contract function, the maximum number of smart-contract-related transactions involving that specific contract in one episode is approximately 50. The experimental parameters are set as shown in Table 3.

For benchmarking, DCSCY is compared with two baseline methods commonly used in sharded blockchain systems. The random-based yanking strategy relocates the smart contract to a randomly selected shard in each episode. This approach is similar to that used in Omniledger [23], where committee or execution assignments are randomized to ensure liveness and fairness. The order-based yanking method, on the other hand, follows a round-robin strategy to sequentially relocate the contract among shards, representing a naive yet deterministic scheduling policy. This approach mimics designs used in early sharded frameworks such as RapidChain [31]. These baselines serve as non-learning references to demonstrate the performance gain achieved by the proposed DRL-based optimization approach.

Table 3. Hyperparameters.

Notation	Description	Value
n_k	The node number of each shard	[5, 20]
K	The total shard number	[3, 7]
α	Hop number for predicting the trajectory of smart contract yanking	[1, 4]
γ	The proposed block in the fix episode	5
$S\_exe_k^\varepsilon$	Max requests processed by the k-th shard during the ε-th episode	50
$S\_gen_k^\varepsilon$	Max transactions generated by the k-th shard during the ε-th episode	50
e	The epoch number	50
λ_1	The weight of $S\_exe_k^\varepsilon$	0.6
λ_2	The weight of $S\_wait_k^\varepsilon$	0.2
λ_3	The weight of $C\_yank_k^\varepsilon$	0.2

5.1. The Experimental Framework

An experimental framework is established utilizing a MacBook Pro laptop equipped with an Apple M3 Max chip, comprising a 16-core CPU, a 40-core GPU, and 128 GB of memory (Apple Inc., Cupertino, CA, USA), to evaluate a proposed blockchain sharding scheme. The experimental environment is configured through the deployment of a virtual machine, leveraging Python 3.8.10 and PyTorch 1.13.1. In this configuration, the discrete DRL algorithm, DQN, is employed to train the model over 50 epochs. The simulation environment is designed to assess the performance, with the number of nodes per shard varying between 5 and 20.

5.2. The Experimental Results

In the experimental evaluations, the performance of the proposed DCSCY and DCSCY-RC frameworks is compared with order-based and random-based baselines. Three key metrics are used for assessment: the reward convergence during training, the total number of smart contract calling requests, and the total waiting time. The latter two metrics also indirectly reflect the system throughput: a higher number of processed smart contract calling requests within a fixed time frame indicates a better throughput, while a shorter total waiting time implies more rapid transaction processing, which similarly contributes to a higher throughput. All experiments are conducted under varying node numbers, shard numbers, and hop distances. DCSCY-RC additionally incorporates Reward Centering (RC) [38] to improve the learning stability by subtracting the empirical average of the observed rewards.

As depicted in Figure 4, the figure illustrates the reward convergence trends for different yanking schemes across 50 training epochs. The x-axis represents the training epochs, while the y-axis represents the reward value obtained during training. The DCSCY-RC approach exhibits the most efficient performance, attaining rapid convergence with higher and more stable reward values after approximately 20 epochs, indicating its enhanced decision-making capability to optimize the smart contract execution. While DCSCY also converges to a high reward, it experiences slightly larger fluctuations during the early training phase, suggesting that the resource coordination mechanism in DCSCY-RC improves the learning efficiency. In contrast, order-based yanking struggles with stability and maintains consistently low rewards, highlighting its inefficiency in scheduling execution. Random yanking exhibited the poorest performance, characterized by larger fluctuations and an inability to converge to a stable strategy, indicating inefficient and unpredictable scheduling. Overall, the results confirm that DRL-based methods outperform the traditional approaches, with DCSCY-RC achieving the best balance between convergence speed and the final reward, making it an effective solution for optimizing smart contract execution in dynamic blockchain environments.

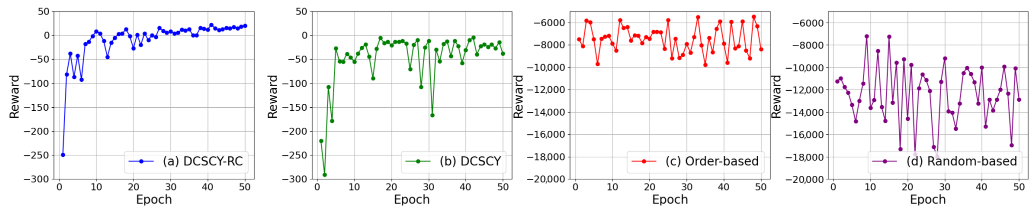

Figure 4. Reward convergence comparison for different yanking schemes. (**a**) DCSCY-RC, (**b**) DCSCY, (**c**) order-based, and (**d**) random-based.

As shown in Figure 5, the *x*-axis represents the node number (left), shard number (middle), and hop number (right), while the *y*-axis represents the length of the queue for the number of smart contract calling requests (log scale). The experimental results demonstrate that DRL-based methods outperform order-based and random-based methods in reducing the length of the queue in the number of smart contract calling requests. Compared to order-based and random-based methods, there are average reductions of 98.45% and 98.32% for different node numbers, 98.01% and 97.79% for different shard numbers, and 97.64% and 98.15% for different hop numbers, respectively. Moreover, DCSCY-RC enhances the performance further compared to DCSCY, achieving reductions of 7.44%, 14.46%, and 6.50% in these conditions, respectively.

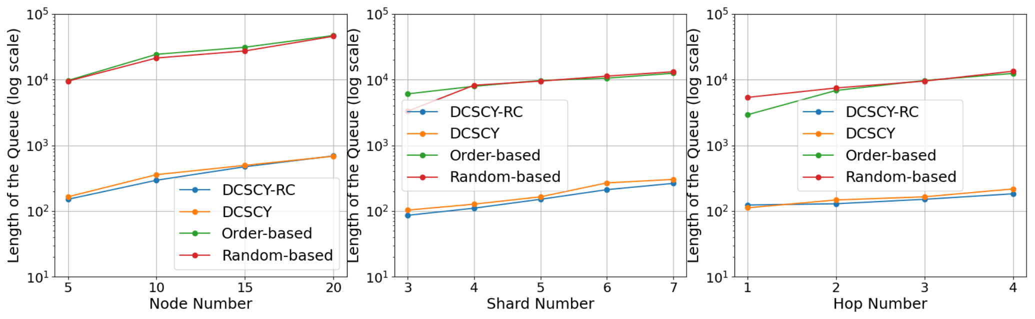

Figure 5. Total smart contract calling request comparison under different conditions.

As shown in Figure 6, the *x*-axis represents node number (left), shard number (middle), and hop number (right), while the *y*-axis represents the total waiting time. The experimental results demonstrate that DCSCY-RC and DCSCY outperform order-based and random-based methods in reducing the total waiting time for smart contract execution. On average, DCSCY reduces the total waiting time by 98.72% and 98.88% compared to that with the order-based and random-based methods for different node numbers, 97.99% and 98.73% for different shard numbers, and 98.53% and 99.05% for different hop numbers, respectively. Moreover, DCSCY-RC enhances the performance further compared to DCSCY, achieving reductions of 16.40% and 22.47% for different node numbers and shard numbers. However, for different hop numbers, the performance of DCSCY-RC is almost identical to that of DCSCY. These results demonstrate the overall effectiveness of DCSCY and the additional benefits of DCSCY-RC.

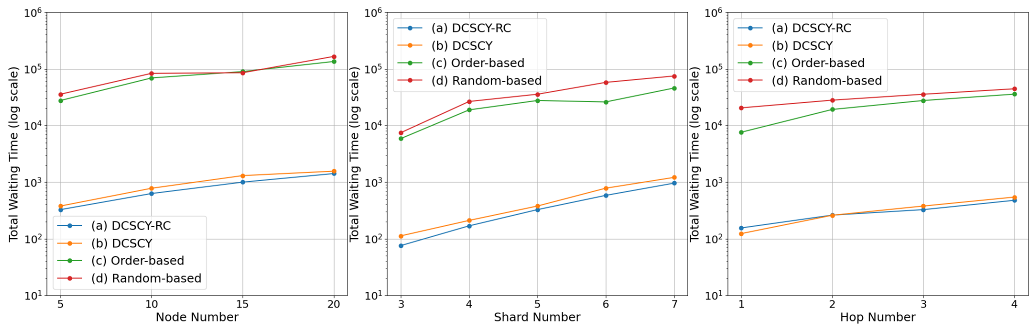

Figure 6. Total waiting time comparison under different conditions.

As shown in Figure 7, the experimental results demonstrate that 1-Hop and 2-Hop achieve faster convergence and better stability, whereas 3-Hop and 4-Hop exhibit greater volatility during the convergence process. As the hop number increases, the number of processed calls rises, resulting in higher reward values. However, an increase in the hop number also intensifies the complexity of predicting the trajectory of the smart contract, thereby posing additional challenges for the DRL model. In all four scenarios from 1-Hop to 4-Hop, the DCSCY-RC method maintains a better convergence performance, indicating that DCSCY-RC can sustain a robust performance even in multi-hop settings.

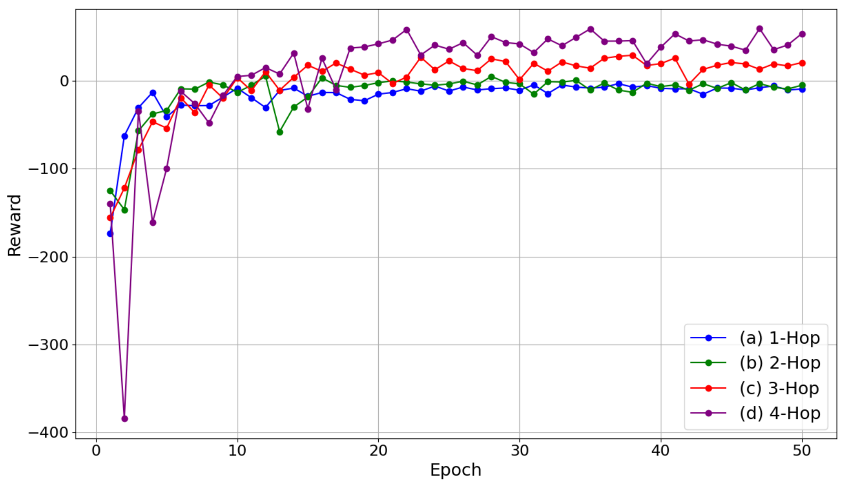

Figure 7. Reward convergence comparison for different hop numbers in the DCSCY-RC smart contract yanking system. (a) 1-Hop, (b) 2-Hop, (c) 3-Hop, and (d) 4-Hop.

6. Conclusions

This paper presented DCSCY, a DRL-Based Cross-Shard Smart Contract Yanking framework designed to improve the execution efficiency in sharded blockchain environments. By formulating contract migration as a sequential decision-making problem, DCSCY dynamically learns the optimal relocation strategies that reduce the transaction latency, minimize node waiting times, and balance yanking costs. The experimental results demonstrated that DCSCY reduced the smart contract waiting times and remaining call requests. Compared to order-based and random-based methods, DCSCY improved the performance by over 95%, effectively enhancing the contract execution efficiency. Additionally, the DCSCY-RC mechanism improves the performance further by approximately 10% compared to DCSCY, enhancing the robustness in sharded blockchains. In addition to its quantitative advantages, DCSCY tackles a key scalability challenge in blockchain systems by facilitating efficient and low-latency execution of high-frequency cross-shard smart contract interactions. Its intelligent contract relocation mechanism mitigates synchronization bottlenecks in locking-based approaches and opens new possibilities for scalable dApps, especially in DeFi, real-time IoT coordination, and supply chain management. This work assumes a non-adversarial environment where the nodes operate honestly and the network is only affected by stochastic delays. In the future, we will consider the possibility that cross-shard environments may face adversarial threats such as denial-of-service (DoS) attacks, malicious queuing behavior, state tampering during contract migration, and Byzantine faults in the shard consensus. Also, we will focus on enhancing DCSCY's practicality and robustness by incorporating realistic gas-cost modeling and adversarial resilience and validating its deployment on real-world sharded platforms.

Author Contributions: Conceptualization, Y.W. and Z.Z.; methodology, Y.W., Z.Z., G.Y. and X.W.; software, Y.W., Z.Z., H.Y. and C.S.; validation, Y.W., Z.Z., H.Y. and C.S.; formal analysis, Y.W., Z.Z., G.Y. and X.W.; investigation, Y.W. and Z.Z.; data curation, Y.W. and Z.Z.; writing—original draft preparation, Y.W. and Z.Z.; writing—review and editing, G.Y. and X.W.; visualization, H.Y.; supervision, G.Y., X.W., W.N., R.P.L. and Z.C.; project administration, G.Y. and X.W.; funding acquisition, Z.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a project of the Ministry of Science and Technology (Grant D20011).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Zheng, Z.; Xie, S.; Dai, H.N.; Chen, X.; Wang, H. Blockchain challenges and opportunities: A survey. *Int. J. Web Grid Serv.* **2018**, *14*, 352–375. [CrossRef]
2. Kayikci, S.; Khoshgoftaar, T.M. Blockchain meets machine learning: A survey. *J. Big Data* **2024**, *11*, 9. [CrossRef]
3. Guo, H.; Yu, X. A survey on blockchain technology and its security. *Blockchain Res. Appl.* **2022**, *3*, 100067. [CrossRef]
4. Soltani, P.; Ashtiani, F. Analytical Modeling and Throughput Computation of Blockchain Sharding. *IEEE Trans. Parallel Distrib. Syst.* **2024**, *35*, 983–997. [CrossRef]
5. Kruglik, S.; Nazirkhanova, K.; Yanovich, Y. Challenges beyond blockchain: Scaling, oracles and privacy preserving. In Proceedings of the 2019 XVI International Symposium "Problems of Redundancy in Information and Control Systems" (REDUNDANCY), Moscow, Russia, 21–25 October 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 155–158.
6. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Decentralized Business Review, 2008, p. 21260. Available online: https://assets.pubpub.org/d8wct41f/31611263538139.pdf (accessed on 8 July 2025).
7. Buterin, V. Ethereum white paper. *GitHub Repos.* **2013**, *1*, 16–23.
8. Buterin, V. A next-generation smart contract and decentralized application platform. *White Pap.* **2014**, *3*, 2-1.
9. Jiang, E.; Qin, B.; Wang, Q.; Wang, Z.; Wu, Q.; Weng, J.; Li, X.; Wang, C.; Ding, Y.; Zhang, Y. Decentralized finance (DeFi): A survey. *arXiv* **2023**, arXiv:2308.05282. [CrossRef]
10. Mohammed Abdul, S.S.; Shrestha, A.; Yong, J. Toward the Mass Adoption of Blockchain: Cross-Industry Insights from DeFi, Gaming, and Data Analytics. *Big Data Cogn. Comput.* **2025**, *9*, 178. [CrossRef]
11. Wang, X.; Yu, G.; Liu, R.P.; Zhang, J.; Wu, Q.; Su, S.W.; He, Y.; Zhang, Z.; Yu, L.; Liu, T.; et al. Blockchain-enabled fish provenance and quality tracking system. *IEEE Internet Things J.* **2021**, *9*, 8130–8142. [CrossRef]
12. Wang, H.; Wang, T.; Shi, L.; Liu, N.; Zhang, S. A blockchain-empowered framework for decentralized trust management in Internet of Battlefield Things. *Comput. Netw.* **2023**, *237*, 110048. [CrossRef]
13. Gadiraju, D.S.; Aggarwal, V. Prism blockchain enabled Internet of Things with deep reinforcement learning. *Blockchain Res. Appl.* **2024**, *5*, 100205. [CrossRef]
14. Yu, G.; Wang, X.; Yu, K.; Ni, W.; Zhang, J.A.; Liu, R.P. Scaling-out blockchains with sharding: An extensive survey. In *Blockchains for Network Security: Principles, Technologies and Applications*; Institution of Engineering and Technology: London, UK, 2020.
15. Zhang, Z.; Yu, G.; Sun, C.; Wang, X.; Wang, Y.; Zhang, M.; Ni, W.; Liu, R.P.; Reeves, A.; Georgalas, N. TbDd: A new trust-based, DRL-driven framework for blockchain sharding in IoT. *Comput. Netw.* **2024**, *244*, 110343. [CrossRef]
16. Liu, Y.; Liu, J.; Salles, M.A.V.; Zhang, Z.; Li, T.; Hu, B.; Henglein, F.; Lu, R. Building blocks of sharding blockchain systems: Concepts, approaches, and open problems. *Comput. Sci. Rev.* **2022**, *46*, 100513. [CrossRef]

17. Dang, H.; Dinh, T.T.A.; Loghin, D.; Chang, E.C.; Lin, Q.; Ooi, B.C. Towards scaling blockchain systems via sharding. In Proceedings of the 2019 International Conference on Management of Data, Amsterdam, The Netherlands, 30 June–5 July 2019; pp. 123–140.
18. Huang, H.; Lin, Y.; Zheng, Z. Account Migration across Blockchain Shards using Fine-tuned Lock Mechanism. In Proceedings of the IEEE INFOCOM 2024-IEEE Conference on Computer Communications, Vancouver, BC, Canada, 20–23 May 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 271–280.
19. Wang, G.; Shi, Z.J.; Nixon, M.; Han, S. Sok: Sharding on blockchain. In Proceedings of the 1st ACM Conference on Advances in Financial Technologies, Zurich, Switzerland, 21–23 October 2019; pp. 41–61.
20. Buterin, V. Cross-Shard Contract Yanking. 2018. Available online: https://ethresear.ch/t/cross-shard-contract-yanking/1450 (accessed on 3 January 2025).
21. Buterin, V. Phase 2 Pre-Spec: Cross-Shard Mechanics. 2019. Available online: https://ethresear.ch/t/phase-2-pre-spec-cross-shard-mechanics/4970 (accessed on 3 January 2025).
22. Robinson, P.; Ramesh, R.; Johnson, S. Atomic crosschain transactions for ethereum private sidechains. *Blockchain Res. Appl.* **2022**, *3*, 100030. [CrossRef]
23. Kokoris-Kogias, E.; Jovanovic, P.; Gasser, L.; Gailly, N.; Syta, E.; Ford, B. Omniledger: A secure, scale-out, decentralized ledger via sharding. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–23 May 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 583–598.
24. Al-Bassam, M.; Sonnino, A.; Bano, S.; Hrycyszyn, D.; Danezis, G. Chainspace: A sharded smart contracts platform. *arXiv* **2017**, arXiv:1708.03778. [CrossRef]
25. Zhang, Z.; Yin, H.; Wang, Y.; Yu, G.; Wang, X.; Ni, W.; Liu, R.P. Enabling Efficient Cross-Shard Smart Contract Calling via Overlapping. In Proceedings of the International Conference on Provable Security, Gold Coast, Australia, 25–27 September 2024; Springer: Berlin/Heidelberg, Germany, 2024; pp. 164–178.
26. Li, M.; Lin, Y.; Zhang, J.; Wang, W. Jenga: Orchestrating smart contracts in sharding-based blockchain for efficient processing. In Proceedings of the 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), Bologna, Italy, 10–13 July 2022; IEEE: Piscataway, NJ, USA, 2022, pp. 133–143.
27. Qi, X.; Li, Y. LightCross: Sharding with Lightweight Cross-Shard Execution for Smart Contracts. In Proceedings of the IEEE INFOCOM 2024-IEEE Conference on Computer Communications, Vancouver, BC, Canada, 20–23 May 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 1681–1690.
28. Liang, J.; Yao, P.; Chen, W.; Hong, Z.; Zhang, J.; Cai, T.; Sun, M.; Zheng, Z. SPARROW: Expediting Smart Contract Execution for Blockchain Sharding via Inter-shard Caching. In *IEEE Transactions on Parallel and Distributed Systems*; IEEE: Piscataway, NJ, USA, 2024.
29. Wels, S. Guaranteed-TX: The Exploration of a Guaranteed Cross-Shard Transaction Execution Protocol for Ethereum 2.0. Master's Thesis, University of Twente, Enschede, The Netherlands, 2019.
30. Al Bassam, M. Securely Scaling Blockchain Base Layers. Ph.D. Thesis, University College London, London, UK, 2020.
31. Zamani, M.; Movahedi, M.; Raykova, M. Rapidchain: Scaling blockchain via full sharding. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 931–948.
32. Yu, G.; Wang, X.; Ni, W.; Lu, Q.; Xu, X.; Liu, R.P.; Zhu, L. Adaptive resource scheduling in permissionless sharded-blockchains: A decentralized multiagent deep reinforcement learning approach. *IEEE Trans. Syst. Man Cybern. Syst.* **2023**, *53*, 7256–7268. [CrossRef]
33. Yang, X.; Xu, T.; Zan, F.; Ye, T.; Mao, Z.; Qiu, T. An Overlapping Self-Organizing Sharding Scheme Based on DRL for Large-Scale IIoT Blockchain. *IEEE Internet Things J.* **2023**, *11*, 5681–5695. [CrossRef]
34. Li, P.; Song, M.; Xing, M.; Xiao, Z.; Ding, Q.; Guan, S.; Long, J. SPRING: Improving the Throughput of Sharding Blockchain via Deep Reinforcement Learning Based State Placement. In Proceedings of the ACM on Web Conference 2024, Singapore, 13–17 May 2024; pp. 2836–2846.
35. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
36. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347. [CrossRef]
37. Etherscan. Etherscan: Ethereum Blockchain Explorer. Available online: https://etherscan.io/ (accessed on 20 February 2025).
38. Naik, A.; Wan, Y.; Tomar, M.; Sutton, R.S. Reward Centering. *arXiv* **2024**, arXiv:2405.09999. [CrossRef]