

Alzo: Auto-Tuning with Reinforcement Learning for DAG-based Blockchains

Qiuyu Ding
dingqiuyu@stu.pku.edu.cn
School of Computer Science
Peking University
Beijing, China

Rongkai Zhang
rkzhang@stu.pku.edu.cn
School of Computer Science
Peking University
Beijing, China

Qinnan Zhang*
zhangqn@buaa.edu.cn
Institute of Artificial Intelligence
Beihang University
Beijing, China

Zhen Xiao†
xiaozhen@pku.edu.cn
School of Computer Science
Peking University
Beijing, China

Jieyi Long
jieyi@thetalabs.org
Theta Labs, Inc.
San Jose, USA

Mingchao Wan
chainmaker@baec.org.cn
Beijing Academy of Blockchain and
Edge Computing
Beijing, China

Sen Liu
liusen@baec.org.cn
Beijing Academy of Blockchain and
Edge Computing
Beijing, China

Jin Dong†
dongjin@baec.org.cn
Beijing Academy of Blockchain and
Edge Computing
Beijing, China

Abstract

As critical infrastructure for Web 3.0, DAG-based blockchains promise high throughput for DeFi, IoT, and DApps. However, realizing this potential is challenging, as system performance is dictated by a multitude of interdependent parameters across network, node, and consensus layers. Manual configuration fails to adapt to dynamic workloads, leading to suboptimal performance. We introduce Alzo, a novel auto-tuner that employs hierarchical reinforcement learning (HRL) to navigate this complex configuration space. By decomposing the DAG blockchain’s workflow into distinct stages, Alzo’s HRL policy learns from stage-level performance metrics to control critical parameters governing consensus, execution, and graph topology in real-time. Furthermore, we employ a shadow-control loop to ensure the safety of all parameter adjustments. Our experiments show that Alzo significantly outperforms other configurations, achieving higher throughput and lower latency under variable workloads with minimal overhead.

CCS Concepts

• **Computer systems organization** → **Distributed architectures**; • **Computing methodologies** → **Reinforcement learning**; • **Software and its engineering** → *Distributed systems organizing principles*.

*Also with Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beihang University.

†Corresponding authors.



This work is licensed under a Creative Commons Attribution 4.0 International License. *WWW '26, Dubai, United Arab Emirates.*

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2307-0/2026/04

<https://doi.org/10.1145/3774904.3792448>

Keywords

Web3.0, Blockchain, DAG Blockchain, Reinforcement Learning

ACM Reference Format:

Qiuyu Ding, Rongkai Zhang, Qinnan Zhang, Zhen Xiao, Jieyi Long, Mingchao Wan, Sen Liu, and Jin Dong. 2026. Alzo: Auto-Tuning with Reinforcement Learning for DAG-based Blockchains. In *Proceedings of the ACM Web Conference 2026 (WWW '26)*, April 13–17, 2026, Dubai, United Arab Emirates. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3774904.3792448>

1 Introduction

As the demand for blockchain scalability intensifies, novel architectures have emerged to overcome the limitations of traditional linear chains. Among these, systems based on Directed Acyclic Graphs (DAGs) have garnered significant academic and industrial attention [39]. By allowing multiple blocks to be proposed concurrently, DAG architectures fundamentally enhance transaction parallelism and offer a promising path towards high-throughput distributed ledgers. Prominent platforms like Conflux [19] are being developed as foundational infrastructure for Web3.0 [22, 32], underscoring the technology’s potential. However, the performance of these complex systems is exquisitely sensitive to their configuration. The vast parameter space means that different tuning choices can lead to orders-of-magnitude differences in throughput and latency. Furthermore, as distributed systems facing real-world, fluctuating workloads, they critically lack the ability to dynamically adapt their configurations to maintain optimal performance.

To address the challenge of finding optimal system parameters, automatic tuning has become an active area of research. Inspired by advances in distributed databases, recent works have employed Deep Reinforcement Learning (DRL) [3, 4, 40] for permissioned blockchain auto-tuning. For instance, AdaChain adaptively selects the best-performing blockchain architecture, but its approach of switching entire architectures can lead to disruptive system restarts

[41]. Athena proposes a multi-agent DRL system to optimize parameters for nodes in Hyperledger Fabric [21]. However, a critical review of existing DRL-based approaches reveals fundamental limitations. They often lack sufficient consideration for workload dynamism, making them ill-suited for online tuning [33, 49]. Moreover, these solutions are predominantly designed for permissioned, linear blockchains and fail to address the unique topological and execution characteristics of DAG-based systems. Crucially, they typically employ a “flat” RL agent that attempts to learn a direct mapping from the entire system state to a high-dimensional, combined parameter space. This approach struggles with the curse of dimensionality and a severe credit assignment problem, leading to inefficient learning and suboptimal policies.

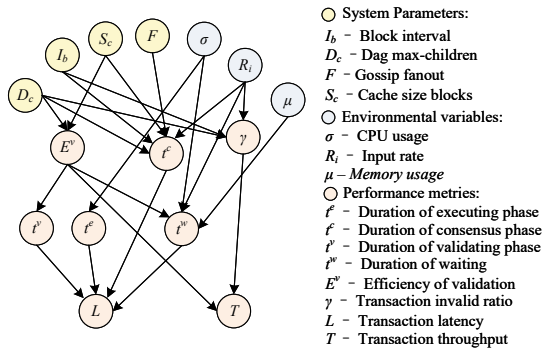


Figure 1: How Parameters Influence System Performance.

However, existing work lacks specialized parameter tuning methods for DAG-based blockchains. Tuning a DAG blockchain presents a unique set of challenges that existing methods are not equipped to handle. The parameter space is not just large; it is deeply interconnected across different system modules. As depicted in Figure 1, adjusting a Ledger parameter like Dag max-children (the number of children references) directly impacts network propagation and node-level validation workload. A monolithic, flat agent cannot effectively model or navigate these intricate, cross-domain trade-offs. Moreover, unconstrained tuning can easily destabilize the system. An excessive gossip_fanout value could saturate network bandwidth, while a large cache_size_blocks might exhaust memory resources. Any practical tuning framework must therefore operate within a “safe envelope,” respecting explicit constraints on resources like CPU, memory, and network I/O. The high degree of concurrency in DAGs complicates performance analysis. When latency increases, it is non-trivial to determine whether the bottleneck lies in tip selection, network gossip, or state persistence. This ambiguity makes it difficult for a single learning agent to assign credit or blame to a specific parameter adjustment.

This research gap highlights the need for a more sophisticated, structured control strategy. Our work, **Alzo**, addresses these shortcomings by introducing a novel hierarchical reinforcement learning framework specifically designed for the autonomous and adaptive tuning of DAG-based blockchains.

To solve the challenges above, **Alzo**’s design is centered on two core principles: hierarchical decomposition and constrained optimization. First, we employ a hierarchical reinforcement learning

architecture [6, 7, 25, 28] that decouples the complex tuning problem into two levels. A high-level Meta-Controller operates on a slower timescale, monitoring the global system state and workload to make strategic decisions. Its actions are not direct parameter changes but rather setting priorities for different system domains (e.g., Network, Ledger, Full-node). These priorities are then passed to low-level, specialized Controllers, which are responsible for the fine-grained tuning of parameters within their specific domain. This hierarchical structure drastically reduces the action space for each agent, clarifies credit assignment by linking performance bottlenecks to specific domains, and enables an interpretable, modular approach to system optimization.

Second, **Alzo** explicitly ensures that parameter adjustments are safe and resource-aware. We introduce a constraint-aware mechanism that integrates operational constraints directly into the agent’s learning objective. By leveraging Lagrangian relaxation [13–15, 18], we transform the constrained optimization problem into an unconstrained one, where the agent is penalized via the reward function for violating predefined parameter bounds. This formulation enables **Alzo** to learn policies that aggressively optimize for performance while proactively avoiding configurations that would lead to system instability or resource violations. Furthermore, we propose a shadow-circuit mechanism to validate parameter safety before deployment, providing a hard guarantee that prevents malicious or erroneous adjustments from compromising the live system.

The primary contributions of this paper are summarized as follows:

- By leveraging hierarchical reinforcement learning with a strategic Meta-Controller and tactical domain-specific Controllers, our framework enables fine-grained, explainable, and adaptive optimization that addresses the unique challenges inherent to DAG-based blockchains.
- We design a safety-aware tuning mechanism that incorporates system constraints into the learning process. We use Lagrangian relaxation to penalize resource violations in the reward function, ensuring that the learned policies are both high-performing and stable.
- We conduct a rigorous experimental analysis on Conflux. Specifically, **Alzo** achieves 50.23% and 54.84% higher throughput, along with 35.93% and 61.50% lower latency compared to DQN-based and BO-based methods, respectively.

2 Background and Related Work

2.1 DAG blockchain

Directed Acyclic Graph (DAG)-based blockchain consensus mechanisms can be categorized based on the fundamental unit proposed by each node for consensus: block-based DAGs and transaction-based DAGs. In block-based DAG architectures, transactions are typically bundled into blocks. For instance, Conflux [19] employs a main chain to guide the topological growth of the DAG, thereby achieving efficient transaction consensus. MorphDAG [47] is designed to address elasticity challenges in storage and transaction processing under variable blockchain workloads. Several other works propose parallel chain architectures where multiple instances or paths produce blocks concurrently. Occam [43] utilizes a DAG

structure featuring exponentially expanding and contracting concurrent paths. OHIE [45] organizes blocks into parallel chains and devises a rank-based sorting algorithm to ensure a total order of transactions. Similarly, Hashgraph [1] and DAG-rider [17] also explore concurrent block production within a DAG framework. Block-based DAG architectures are prevalent due to their structured approach, which can simplify certain aspects of consensus and state management.

Transaction-based DAGs, alternatively, allow individual transactions to be attached to existing vertices (previous transactions) according to a deterministic algorithm. This fine-grained approach at the transaction level offers opportunities for customized optimizations, such as incorporating temporal characteristics for transactions, and often results in lower operational costs. Consequently, transaction-based DAGs have found notable applications in resource-constrained environments like the Internet of Things (IoT). Notable examples in this category include Spectre [35] and Phantom [36]. IOTA [34] utilizes a weighted random walk algorithm, specifically Markov Chain Monte Carlo (MCMC), for new block to select two preceding tips for approval. TidyBlock [29] introduces a novel consensus mechanism incorporating a transaction ordering algorithm and a block selection mechanism to enhance the efficiency of DAG blockchains in IoT, catering to varying transaction priorities and waiting times. RT-DAG [23] improves address resource utilization by assigning priorities to transactions and introducing an Address Resource Graph (ARG) to capture address usage during execution. FLUID [27] designs a transaction dependency tracking structure ensuring that consecutive, dependent transactions are processed in the correct sequence.

2.2 Blockchain Parameters Tuning

Automatic configuration tuning is a well-studied problem in distributed systems, with existing works generally categorized into search-based, machine learning-based, and reinforcement learning-based methods. Search-based methods, particularly Bayesian Optimization (BO), have been effectively used for black-box optimization but often suffer from long execution times and an inability to leverage historical experience [2, 8, 12]. Machine learning-based approaches build predictive models to map parameters to performance, yet they often require costly retraining in new environments and are sensitive to sample quality [5, 37]. More recently, reinforcement learning (RL) has emerged as a powerful paradigm, treating tuning as a trial-and-error process that can effectively reduce training costs and adapt to changing environments [48].

While permissioned blockchains share characteristics with distributed databases, the aforementioned methods cannot be directly applied due to the unique complexities introduced by consensus mechanisms and intricate inter-parameter dependencies. Inspired by these advances, recent research has employed Deep Reinforcement Learning (DRL) for permissioned blockchain auto-tuning. For instance, AdaChain adaptively selects the best-performing blockchain architecture under dynamic workloads, but its approach of switching architectures can lead to disruptive system restarts [41]. Other works have focused on optimizing specific system aspects, such as dynamically adjusting block size [33], selecting consensus parameters [24], or optimizing sharding strategies [46].

Among the most relevant works, Athena proposes a multi-agent DRL system to optimize heterogeneous parameters for different nodes in Hyperledger Fabric [21]. However, a critical review of existing DRL-based approaches, including Athena and others [33, 49], reveals several fundamental limitations. First, they often lack sufficient consideration for the dynamism of workloads, making them ill-suited for online tuning in non-stationary environments. Second, they frequently fail to account for the significant performance degradation caused by invalid transactions arising from conflicts, a critical issue in real-world deployments.

3 System Design

3.1 System Model

Alzo functions as a parameter tuning module attached to each node in the Conflux network. Each node deploys an independent tuning agent, leveraging the inherent property of DAG-based blockchains that do not require uniform parameter settings across the network. Alzo adopts the same system model as Conflux, operating under network conditions characterized by an uncertain yet bounded upper limit Δ , which dictates the maximum permissible delay for message transmission throughout the network. While this bound is not concretely specified, it is instrumental in governing the network's functional dynamics. Consequently, the addition of Alzo's parameter tuning module does not compromise the system's liveness or safety properties. To ensure the effectiveness of parameter adjustments, we propose specific mechanisms detailed in subsequent sections.

During deployment, Alzo operates on a discrete time-step basis, where each time step corresponds to every n confirmed blocks. The tuning agent is triggered at each time step to generate recommended configuration parameters. The empirical value of n is specified in the Appendix.

3.2 Parameters and Performance

The foundational step in developing an autonomous tuning agent is to establish a formal model that connects the system's configurable parameters to its observable performance. We evaluate the system's efficacy using the two primary top-level metrics in distributed ledgers: **latency** and **throughput**. To enable fine-grained control for our hierarchical agent, we dissect the end-to-end transaction latency into a series of distinct processing stages that align with the DAG transaction lifecycle.

The end-to-end latency for a single transaction is defined as the difference between its confirmation time (T_c) and its submission time (T_s):

$$\text{Latency} = T_c - T_s \quad (1)$$

To gain deeper insight into performance bottlenecks, we decompose this total latency into seven sequential and concurrent stages:

$$\text{Latency} = T_{\text{prep}} + T_{\text{preSel}} + T_{\text{preVal}} + T_{\text{cons}} + T_{\text{ord}} + T_{\text{conf}} + T_{\text{state}} \quad (2)$$

where T_{prep} is the transaction preparation and packaging time, T_{preSel} is the pre-block selection time (i.e., choosing tips), T_{preVal} is the pre-block validation time, T_{cons} is the consensus and propagation time, T_{ord} is the ordering and conflict-pruning time, T_{conf} is the confirmation time required to meet finality rules, and T_{state} is the

state update and persistence time. This fine-grained decomposition is crucial as it allows our hierarchical agent, Alzo, to attribute performance bottlenecks to specific stages.

The performance of a DAG-based DLS(Distributed Ledger System) is dictated by the intricate interplay of numerous parameters. We categorize these parameters into three distinct domains to better structure the action space for our control agent: **Full-Node(F)**, **Network(N)**, and **Ledger(L)**. A representative subset of these critical parameters is shown in Table 1.

Table 1: Representative Configurable Parameters in a DAG-based DLS

Parameter	Cat.	Description
mempool_limit	F	Pending transaction pool threshold
cache_size_blocks	F	Pre-block/ledger cache size
db_write_batch	F	Database write batch size
max_blocks_per_fetch	N	Maximum blocks fetched per network request
gossip_fanout	N	Number of gossip peers per message
conn_buffer_size	N	Per-connection send/receive buffer size
msg_size_limit	N	Maximum message batch size
min_round_delay	L	Minimum interval between rounds
pre_Blocks	L	Number of parent references (out-degree)
max_Children	L	Maximum direct children (in-degree)
milestone_interval	L	Confirmation trigger frequency
block_size	L	max transaction number in each block

The relationship between the complete parameter vector $P = [p_1, p_2, \dots, p_n]^T$ and the system's performance is a complex, non-linear black-box function, f :

$$\text{Performance} = f(P) = (\text{Latency}_{\text{avg}}, \text{Throughput}) \quad (3)$$

This function is subject to the hard constraints imposed by the valid parameter ranges:

$$\begin{bmatrix} \ell_1 \\ \ell_2 \\ \vdots \\ \ell_n \end{bmatrix} \leq \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix} \leq \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \quad (4)$$

The overarching goal is to find a parameter vector P^* that solves a multi-objective optimization problem: maximizing throughput while minimizing average latency. Effectively, our system aims to navigate this high-dimensional parameter space to operate along the Pareto frontier of this trade-off, dynamically adapting its configuration to maintain optimal performance under variable workloads. This complex, stage-dependent control problem motivates our design of a hierarchical reinforcement learning framework.

3.3 RL Design

A standard, "flat" reinforcement learning agent faces significant challenges when tuning DAG blockchain parameters due to the curse of dimensionality from the large, combined parameter space, the difficulty in credit assignment, and the need to manage conflicting objectives like throughput and latency.

To overcome these challenges, we adopt a **hierarchical decoupling** strategy. We decompose the optimization problem into a high-level strategic task managed by a **Meta-Controller** and several low-level, domain-specific execution tasks handled by specialized **Controllers**. As shown in Figure 2, each Controller operates in a smaller, domain-specific action space, which simplifies the learning task and accelerates convergence. The Meta-Controller is rewarded for setting effective long-term goals, while each Controller is rewarded for executing its specific strategy, making the learning signals more precise. The Meta-Controller operates on a slower timescale, learning long-term strategies from aggregated performance trends. The Controllers handle rapid, tactical adjustments at a faster timescale, reacting to immediate system feedback.

We formulate this problem as a two-level Markov Decision Process (MDP), which provides a formal framework for our hierarchical approach.

3.3.1 High-Level MDP: The Meta-Controller. The Meta-Controller operates on a macroscopic scale, focusing on overall system health and strategic adaptation to workload dynamics. Its objective is to allocate optimization priorities across different system domains rather than manipulating specific parameters. Its MDP is defined as $(S_h, \mathcal{A}_h, \mathcal{R}_h, \mathcal{P}_h)$, where \mathcal{P}_h is the transition probability function.

State Space (S_h): The state $s_{h,t}$ is a vector capturing external workload characteristics and global Key Performance Indicators (KPIs). The external workload characteristics include the transaction arrival rate ($T_{in,t}$), mempool remaining size ($M_{rs,t}$), and mempool growth rate ($M_{gr,t}$). The global KPIs include normalized throughput ($Th_{n,t}$), latency (L_t), and a binary vector of resource violation flags $c_t \in \{0, 1\}^M$ indicating overload on CPU, memory, network, or disk.

$$s_{h,t} = [T_{in,t}, M_{rs,t}, M_{gr,t}, c_t] \quad (5)$$

Action Space (\mathcal{A}_h): The action $\mathbf{a}_{h,t}$ is a 3-dimensional domain emphasis vector $\mathbf{a}_{h,t} = [a_{N,t}, a_{L,t}, a_{F,t}]$, representing the strategic priorities for the Network (N), Ledger (L) and Full-Node (F) domains. It satisfies $\sum a_{i,t} = 1$ and $a_{i,t} \geq 0$, enforced via a softmax function. For instance, an action [0.5, 0.3, 0.2] would direct the system to prioritize network and ledger performance.

Reward Function (\mathcal{R}_h): The reward is computed over a long time window to evaluate the strategic effectiveness of the Meta-Controller, balancing performance enhancements with system stability:

$$\begin{aligned} r_{h,t} = & \lambda_{\text{thpt}} \cdot \log(1 + \text{Throughput}(s_{h,t}, \mathbf{a}_{h,t})) \\ & - \lambda_{\text{lat}} \cdot \text{Latency}(s_{h,t}, \mathbf{a}_{h,t}) \\ & - \lambda_{\text{smooth}} \cdot \|\mathbf{q}_t - \mathbf{q}_{t-1}\|_2^2 \end{aligned} \quad (6)$$

Here, the logarithmic term on throughput encourages stable, incremental improvements, while penalties are imposed for latency and abrupt strategic shifts (measured by the L2 norm $\|\mathbf{q}_t - \mathbf{q}_{t-1}\|_2^2$), thereby preventing system oscillations. The hyperparameters $\lambda_{(\cdot)}$ are tuned to balance these competing objectives.

State Transitions(\mathcal{P}_h) The state transition function models how the environment responds to actions and captures the temporal characteristics of incoming transactions. In an MDP, the function $P(s_{t+1}|s_t, \mathbf{a}_t)$ defines the probability distribution over the next state s_{t+1} given the current state s_t and action \mathbf{a}_t .

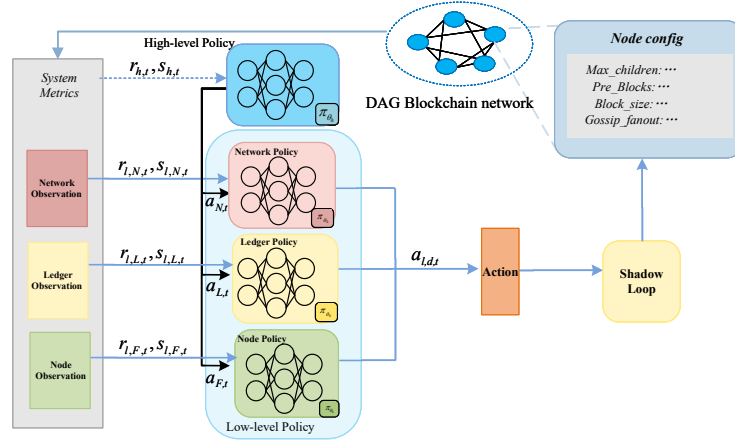


Figure 2: System Overview of Alzo.

In this framework, the state transition is determined jointly by the agent’s actions and the environment. Specifically, the results of parameter adjustments ultimately form the new state s_{t+1} . This transition process naturally satisfies the Markov property and can be formalized as:

$$s_{t+1} = f(s_{h,t}, \mathbf{a}_{h,t}, \xi_t), \quad (7)$$

where f denotes the transition function implemented by the blockchain transaction processing engine, and ξ_t represents environmental randomness, such as the stochastic arrival of transactions and non-determinism in consensus.

3.3.2 Low-Level MDPs: The Domain Controllers. Each Controller $d \in \{N, L, F\}$ acts as a tactical expert, receiving strategic guidance from the Meta-Controller and optimizing its local domain through precise parameter adjustments. Its MDP is defined as $(S_{l,d}, \mathcal{A}_{l,d}, \mathcal{R}_{l,d}, \mathcal{P}_{l,d}, \gamma_l)$.

State Space ($S_{l,d}$): The low-level state space $S_{l,d}$ integrates high-level directives with detailed local telemetry, enabling informed decision-making. Specifically, the state $s_{l,d,t}$ includes the domain-specific weight $a_{d,t}$ assigned by the Meta-Controller, which indicates the strategic priority of the domain, along with local metrics tailored to each controller. For the Network Controller ($d = N$), the state is formulated as $s_{l,N,t} = [B_{u,t}, Q_{len,t}, a_{N,t}]$, where $B_{u,t}$ simplifies bandwidth utilization, $Q_{len,t}$ represents message queue lengths, $a_{N,t}$ is the domain weight. For the Ledger Controller ($d = L$), the state is $s_{l,L,t} = [T_{tip,t}, a_{L,t}, O_{rate,t}, C_{usage,t}]$, incorporating tip count ($T_{tip,t}$), domain weight ($a_{L,t}$), orphan block rate ($O_{rate,t}$), and CPU usage ($C_{usage,t}$). For the full-node Controller ($d = F$), the state is $s_{l,F,t} = [a_{F,t}, M_{usage,t}]$, including domain weight ($a_{F,t}$), throughput ($Th_{n,t}$), and memory usage ($M_{usage,t}$). This combination ensures that each Controller has a comprehensive view of both global strategy and local conditions.

Action Space ($\mathcal{A}_{l,d}$): The action space $\mathcal{A}_{l,d}$ consists of normalized adjustments to parameters within its domain. The agent outputs a vector $\mathbf{a}_{l,d,t}$ with elements in $[-1, 1]$, which is mapped to relative changes (e.g., $\pm 10\%$) or absolute values for system parameters. This relative adjustment mechanism promotes stable learning by avoiding abrupt shifts that could destabilize the system.

Reward Function ($\mathcal{R}_{l,d}$): The reward function $\mathcal{R}_{l,d}$ aligns local actions with global objectives, resolves credit assignment challenges, and incorporates Lagrangian constraints to enforce resource limits. It is structured as:

$$r_{l,d,t} = \alpha \cdot r_{local,d,t} + \beta \cdot a_{d,t} \cdot \Delta T_t - \lambda'_{smooth} \|\Delta \mathbf{p}_t\|_2^2 - \sum_k \mu_k \max(0, g_k(\mathbf{p}_t) - b_k) \quad (8)$$

where Component 1 ($\alpha \cdot r_{local,d,t}$) provides a scaled, domain-specific reward to encourage local optimizations within each domain d , with $\alpha > 0$ balancing local objectives with global priorities. Domain-specific rewards include $r_{local,N,t} = T_{preSel,t} + T_{preVal,t}$ for Network domain (pre-selection and pre-validation throughput), $r_{local,L,t} = T_{cons,t}$ for Ledger domain (consensus throughput), and $r_{local,F,t} = T_{state,t} + T_{order,t}$ for Storage domain (state and order access throughput). Component 2 ($\beta \cdot a_{d,t} \cdot \Delta T_t$) rewards actions that yield proportional improvements in overall system throughput, where $\mathbf{a}_{d,t}$ is the action taken by domain d , $\Delta T_t = T_t - T_{t-1}$ represents the throughput change, and $\beta > 0$ weights the importance of throughput enhancement. Component 3 ($\lambda'_{smooth} \|\Delta \mathbf{p}_t\|_2^2$) penalizes parameter ($\Delta \mathbf{p}_t = \mathbf{p}_t - \mathbf{p}_{t-1}$), with coefficient $\lambda'_{smooth} > 0$ promoting configuration stability. Component 4 ($\sum_k \mu_k \max(0, g_k(\mathbf{p}_t) - b_k)$) penalizes parameter violations, where $g_k(\mathbf{p}_t)$ represents the k -th parameter constraint function, b_k is the corresponding threshold, and μ_k are Lagrangian multipliers enforcing operational safety bounds.

Component 4 is The Lagrangian constraints are generated through a dual optimization process to handle inequality constraints within the RL framework. Specifically, for each constraint $g_k(\mathbf{p}_t) \leq b_k$ (e.g., a parameter value within its artificially imposed range), a multiplier μ_k is introduced. During training, these multipliers are updated via dual ascent: if a constraint is violated ($g_k > b_k$), μ_k is increased (e.g., $\mu_k \leftarrow \mu_k + \eta(g_k - b_k)$ with step size η); if satisfied, μ_k can decrease. This iterative process tightens the constraints over time, embedding them directly into the reward function to guide the policy toward feasible solutions without requiring explicit constraint satisfaction at every step. By dynamically adjusting μ_k , the framework effectively balances optimization objectives with artificially imposed parameter range limits, enhancing the overall robustness

and applicability in resource-constrained blockchain environments. **State Transitions** ($\mathcal{P}_{l,d}$) are detailed in the Appendix.

3.4 Integration with DAG Blockchain: Hierarchical Soft Actor-Critic

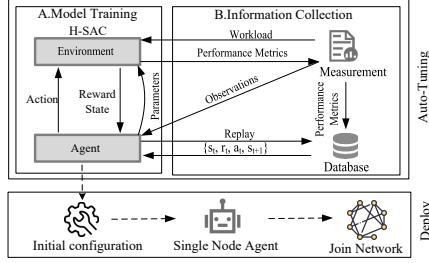


Figure 3: Training process of Alzo.

To solve the two-level MDP, we implement a Hierarchical Soft Actor-Critic (H-SAC) framework. We chose SAC [16, 26] for its sample efficiency, stability, and robust exploration capabilities driven by entropy maximization.

As illustrated in Figure 3, the training process operates as follows: Initially, a baseline configuration is manually specified, with each node in the network running an identical model. During training, states are observed from a benchmark suite where Prometheus is deployed to collect averaged metrics of CPU, memory, and bandwidth load across nodes. After SAC generates actions based on these observations, they are fed into the DAG blockchain environment for execution. The model outputs parameter adjustments that are uniformly applied to the configuration files of all nodes. The training leverages off-policy replay buffers to enhance sample efficiency. Since SAC is an off-policy algorithm, it can learn from pre-collected data sourced from historical blockchain scenarios, encompassing varying transaction loads, network delays, and resource constraints. This historical data is stored in replay buffers, enabling the agents to learn from diverse past experiences and rapidly reach an operational state without requiring extensive real-time interaction with the live blockchain environment.

Training occurs in a nested loop to reflect the hierarchy. The Meta-Controller selects an action q_t and maintains it for a fixed window of k low-level steps. During this window, the Controllers act at each step, collecting experiences (state, action, reward, next_state) in their respective replay buffers—augmented by the shadow loop for constraint validation. At the end of the window, the aggregated high-level reward $R_{h,t}$ is used to train the Meta-Controller, enabling it to learn which strategic goals yield optimal long-term performance. The Controllers are trained continuously using their own experiences and the goal-conditioned reward function. This hierarchical structure effectively resolves credit assignment by attributing global performance changes to the Meta-Controller’s strategic decisions and local improvements to the Controllers’ tactical adjustments.

Moreover, to guarantee that parameter adjustments do not exceed predefined ranges, we incorporate a **shadow loop** mechanism

that acts as a safety layer during both training and inference. Before applying any proposed action (parameter update) from the Controllers, the system simulates the adjustment in a lightweight shadow environment—a virtual replica of the blockchain node that mirrors current states but operates in isolation. If the simulated outcome violates constraints (e.g., a parameter p_i exceeds its valid range $[p_{\min}, p_{\max}]$), the action is clipped to the nearest boundary value to maintain safety. This shadow loop runs in parallel with the main execution, ensuring real-time validation without introducing latency to the primary blockchain processes. During training, experiences from shadow simulations are selectively added to the replay buffers to reinforce boundary-aware learning, further embedding safety constraints into the policy optimization process.

4 Experiments

To evaluate the efficacy and adaptability of our proposed hierarchical reinforcement learning framework, Alzo, we conduct a series of comprehensive experiments. This chapter details the experimental setup, baselines for comparison, and an in-depth analysis of the results.

4.1 Experimental Setup

We selected Conflux as our experimental platform, which has been widely adopted in many research papers. We use `conflux-rust` [10] for client-transaction generation and nodes for validation and confirmation.

4.1.1 Environment Setting and Workload Generation. Our experimental setup consists of 32 physical machines, each equipped with Intel Xeon Platinum CPU cores and 16 GB of RAM. Each physical machine independently runs a blockchain node. Each node is equipped with a client node that issues transactions at a fixed rate. The bandwidth of all network connections between nodes is set to 50 Mbps.

Dynamic Workload: To simulate real-world network fluctuations, we employ a widely adopted transaction workload based on the SmallBank [11] benchmark. The benchmark driver preloads the blockchain with 10k users, each with two accounts. We set N_{hot} of them as hot accounts. When firing transactions, the client randomly picks one of the five modifying transactions with probability P_w and the read-only transactions with probability $1 - P_w$. Each transaction has a certain probability to access the hot accounts, as controlled by the P_{hot} parameter. The client continuously fires N_{trans} transactions every T_{fire} milliseconds. To simulate computation-heavy transactions, each transaction has a $T_{compute}$ interval after it fetches the required world states from the key-value store and before its subsequent operation. To systematically evaluate performance under varying conditions, we define four distinct workload scenarios based on the SmallBank benchmark and real-world data:

- **C1 (Real-World Dataset):** To capture authentic fluctuations, this scenario replays a dataset of 800,000 real transactions collected on Conflux. The workload directly reflects surveyed real-world patterns, including irregular spikes and diverse computation demands.
- **C2 (Low Load, SmallBank-based):** This scenario simulates a low-intensity environment using the SmallBank benchmark with reduced transaction volume. We set $N_{trans} = 3000$

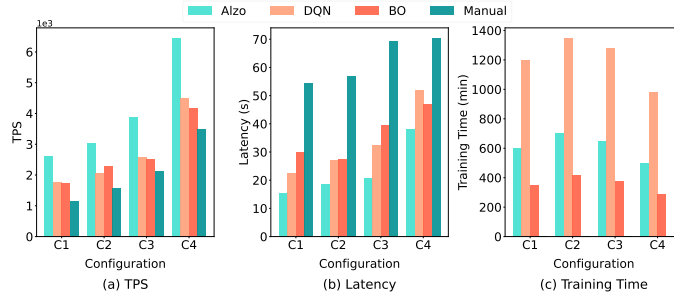


Figure 4: Throughput vs. Latency for Alzo and baseline methods on the Conflux platform.

transactions every $T_{fire} = 1000$ ms, $P_w = 0.2$ for a low write probability, and $P_{hot} = 0.1$ for minimal hot account contention. Computation time is set to $T_{compute} = 2$ ms to represent lightweight transactions. This mimics underutilized networks with sparse activity.

- **C3 (Balanced Load, SmallBank-based):** A medium-intensity workload via SmallBank, balancing read and write operations. Parameters include $N_{trans} = 2000$ every $T_{fire} = 500$ ms, $P_w = 0.5$ for equal read/write mix, $P_{hot} = 0.2$ for moderate contention, and $T_{compute} = 20$ ms. This represents steady-state, everyday network loads.
- **C4 (High Load, SmallBank-based):** This high-intensity scenario uses SmallBank to model resource-strained conditions. We configure $N_{trans} = 700$ every $T_{fire} = 100$ ms, $P_w = 0.7$ for write-heavy operations, $P_{hot} = 0.5$ for high contention, and $T_{compute} = 100$ ms to simulate computation-heavy transactions. This emulates peak-hour or congested network environments.

These conditions allow us to assess each method’s robustness across controlled simulations (C2–C4) and realistic, dynamic scenarios (C1).

4.1.2 *Baselines for Comparison.* We compare Alzo against three baseline methods representing different tuning philosophies:

- **Manual Tuning:** We use static parameter configurations set by human experts.
- **DQN:** A classic reinforcement learning algorithm [20, 31, 44]. The DQN agent learns to map the system state to an action that adjusts a single parameter at a time. We discretize each parameter into 7 bins, resulting in an action space of size 12×7 for 12 tunable parameters (each action selects one parameter and one adjustment level). This baseline demonstrates the challenges of dimensionality and credit assignment that our hierarchical approach is designed to solve.
- **Bayesian Optimization (BO):** A powerful and sample-efficient black-box optimization technique [9, 30, 38, 42]. BO builds a probabilistic model of the objective function and uses an acquisition function to select the most promising parameters to evaluate next. It represents a strong non-RL baseline but is inherently less suited for real-time adaptation in dynamic environments.

4.2 Performance Evaluation

We structure our evaluation around four key research questions (RQs) to assess Alzo’s overall performance, adaptability, the effectiveness of its hierarchical design, and its operational costs. We designed four groups of experiments to answer the following questions:

RQ1: How does Alzo compare against baselines in simultaneously optimizing for throughput and latency?

RQ2: How effectively does Alzo adapt to dynamically changing network loads compared to the baselines?

RQ3: How does the hierarchical structure of Alzo contribute to its performance? Is the high-level strategic coordination necessary?

RQ4: How high are the operational costs and bottlenecks of running Alzo?

4.2.1 *Overall Performance Trade-off (RQ1).* To address RQ1, we evaluate each method’s ability to navigate the trade-off between throughput and latency. We measure TPS, latency, and required training time under four distinct workload conditions: C1, C2, C3, and C4.

As illustrated in Figure 4, Alzo outperforms all baselines across the evaluated conditions. The Manual configurations yield the lowest performance due to their inherent uncertainties and lack of optimization, resulting in static setups that fail to adapt to varying workloads. Specifically, Alzo achieves 50.23%, 54.84%, and 122.60% higher throughput compared to DQN, BO, and Manual, respectively. Additionally, Alzo’s latency is 35.93%, 61.50%, and 71.82% lower than that of DQN, BO, and Manual, respectively. The DQN agent, hindered by its high-dimensional action space and inefficient exploration, achieves only suboptimal balances and underperforms BO in certain scenarios. Bayesian Optimization can identify several effective configurations but is less efficient in exploring the full trade-off space compared to Alzo’s goal-conditioned learning, particularly failing to achieve low latency at high throughput levels.

Regarding training time, Alzo’s hierarchical design significantly reduces convergence time compared to DQN. In contrast, BO requires only approximately 400 minutes of empirical sampling to train the optimizer effectively and obtain reasonable parameter configurations. While manual tuning incurs no training overhead, its lack of flexibility limits its applicability in dynamic environments. Alzo’s superior performance arises from its hierarchical structure: the Meta-Controller strategically sets high-level trade-offs, while the domain-specific Controllers execute these strategies through

fine-grained, coordinated adjustments in a reduced action space. This approach enables more effective and stable optimization.

4.2.2 Adaptability to Dynamic Workloads (RQ2). To address RQ2, we evaluate each method’s ability to adapt to fluctuating conditions using the dynamic workload profile described in Section 4.1.1. An ideal agent should dynamically respond to these changes to minimize performance degradation.

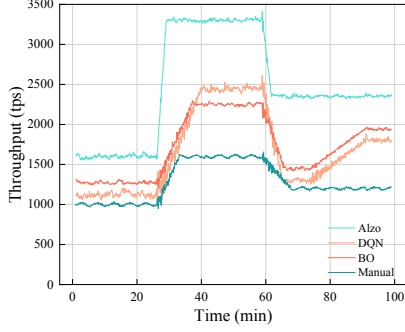


Figure 5: System performance under dynamic workload.

Figure 5 highlights Alzo’s key strength: superior adaptability. During the ramp-up phase, Alzo rapidly adjusts to high loads, identifying and achieving optimal TPS levels with minimal disruption. In contrast, Bayesian Optimization and DQN, which are slower to learn and adapt, can maintain relatively good TPS under stable conditions but exhibit inferior adaptation. The Manual tuning, with its fixed parameter settings, fails to adapt to throughput variations, resulting in suboptimal performance and inability to fully leverage the system’s potential under dynamic workloads. Alzo’s hierarchical design enables this responsiveness, as the Meta-Controller anticipates load shifts and guides the lower-level Controllers to make proactive, coordinated adjustments, ensuring consistent performance even in volatile environments.

4.2.3 Effectiveness of Hierarchical Control (RQ3). To address RQ3 and validate the core design principle of hierarchical coordination, we conduct an ablation study focusing on the Meta-Controller’s role. We compare the full Alzo agent against an ablated version, **Alzo-Flat**, where the Meta-Controller is removed. In Alzo-Flat, a single unified Controller directly manages all configuration parameters, attempting to optimize a global reward signal without hierarchical coordination. To quantify the impact, we present TPS-latency curves for both variants.

The results, presented in Figure 6, confirm the criticality of the hierarchical design. Alzo achieves 27.74% higher throughput and 30.43% lower latency compared to Alzo-Flat. Without the Meta-Controller’s guidance, the single Controller struggles to effectively balance the complex interdependencies among multiple parameters, such as simultaneously optimizing block size, parallelism strategies, and resource allocation. This results in suboptimal decisions, internal bottlenecks, and degraded overall performance. This ablation study underscores how Alzo’s hierarchical structure enhances coordination, efficiency, and adaptability in complex, dynamic environments.

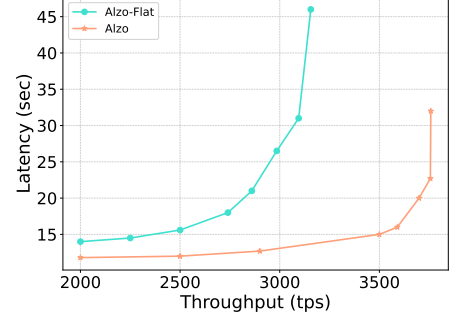


Figure 6: Performance comparison between the full Alzo agent and the ablated Alzo-Flat, showing TPS-latency curves.

4.2.4 Overhead of Learning (RQ4). To address RQ4, we evaluate the computational overhead, focusing on runtime efficiency. While superior performance is crucial, an effective auto-tuning framework must also balance costs to ensure practicality in real-world deployments. We analyze these overheads using data from the experiments in Section 4.1.1, comparing Alzo performance across the four workload conditions (C1–C4). Runtime speed refers to the inference latency during operation, which we measure in dynamic tests.

Table 2: CPU and Memory Resource Usage

Metrics	C1	C2	C3	C4
CPU (ms)	14	13	14	16
Memory (MB)	1182	1152	1188	1233

The experimental results, summarized in Table 2, demonstrate that Alzo exhibits low inference overhead across various workloads, maintaining low latency and acceptable memory consumption.

5 Conclusion

We present Alzo, a hierarchical reinforcement learning framework for automatic parameter tuning in DAG-based blockchains. Alzo uses a Meta-Controller for strategic planning and specialized Controllers for fine-grained adjustments, with a safety-aware shadow loop ensuring stability. Experiments show Alzo achieves 50.23%, 54.84% higher throughput and 35.93%, 61.50% lower latency compared to DQN-based and BO-based method, while rapidly adapting to dynamic workloads with low computational cost. Alzo represents a significant advancement in autonomous, adaptive, and safe configuration for next-generation distributed ledger technologies.

6 Acknowledgments

The authors would like to thank the anonymous reviewers for their comments. This work was supported by the National Key R&D Program of China under Grant 2023YFB2703800, the Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, and the National Natural Science Foundation of China (NSFC) under Grant No. 62372493. The contact authors are Zhen Xiao and Jin Dong.

References

- [1] Zuhair Akhtar. 2019. From blockchain to hashgraph: distributed ledger technologies in the wild. In *2019 International Conference on Electrical, Electronics and Computer Engineering (UPCON)*. IEEE, 1–6.
- [2] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. CherryPick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 469–482.
- [3] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866* (2017).
- [4] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep reinforcement learning: A brief survey. *IEEE signal processing magazine* 34, 6 (2017), 26–38.
- [5] Lin Bao, Xiaohui Liu, Zhipeng Xu, and Bowen Fang. 2018. Autoconfig: Automatic configuration tuning for distributed message systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 29–40.
- [6] Andrew G Barto and Sridhar Mahadevan. 2003. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems* 13, 4 (2003), 341–379.
- [7] Matthew Michael Botvinick. 2012. Hierarchical reinforcement learning and decision making. *Current opinion in neurobiology* 22, 6 (2012), 956–962.
- [8] Stefano Cereda, Stefano Valladares, Paolo Cremonesi, and Stefano Doni. 2021. Cgptuner: a contextual gaussian process bandit approach for the automatic tuning of it configurations under varying workload conditions. *Proceedings of the VLDB Endowment* 14, 8 (2021), 1401–1413.
- [9] Hyunghun Cho, Yongjin Kim, Eunjung Lee, Daeyoung Choi, Yongjae Lee, and Wonjong Rhee. 2020. Basic enhancement strategies when using Bayesian optimization for hyperparameter tuning of deep neural networks. *IEEE access* 8 (2020), 52588–52608.
- [10] Conflux Network. 2024. Conflux-rust: Rust implementation of Conflux protocol. <https://github.com/Conflux-Chain/conflux-rust>. Accessed: 2024-01-15.
- [11] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. Blockbench: A Framework for Analyzing Private Blockchains. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data (SIGMOD '17)*. ACM, New York, NY, USA, 1085–1100. doi:10.1145/3035918.3064033
- [12] Ayat Fekry, Lucian Carata, Thomas Pasquier, Andrew Rice, and Andy Hopper. 2020. To Tune or Not to Tune? In Search of Optimal Configurations for Data Analytics. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (Virtual Event, CA, USA) (KDD '20)*. Association for Computing Machinery, New York, NY, USA, 2494–2504.
- [13] Marshall L Fisher. 1981. The Lagrangian relaxation method for solving integer programming problems. *Management science* 27, 1 (1981), 1–18.
- [14] Marshall L Fisher. 1985. An applications oriented guide to Lagrangian relaxation. *Interfaces* 15, 2 (1985), 10–21.
- [15] Marshall L Fisher. 2004. The Lagrangian relaxation method for solving integer programming problems. *Management science* 50, 12_supplement (2004), 1861–1871.
- [16] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. Pmlr, 1861–1870.
- [17] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. 165–175.
- [18] Claude Lemaréchal. 2001. Lagrangian relaxation. In *Computational combinatorial optimization: optimal or provably near-optimal solutions*. Springer, 112–156.
- [19] Chenxing Li, Peilun Li, Dong Zhou, Zhe Yang, Ming Wu, Guang Yang, Wei Xu, Fan Long, and Andrew Chi-Chih Yao. 2020. A decentralized blockchain with high throughput and fast confirmation. In *Proceedings of the 2020 USENIX Annual Technical Conference*. 515–528.
- [20] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2118–2130.
- [21] Mingzhe Li, Yuxuan Wang, Sitong Ma, Ce Liu, Ding Huo, Yu Wang, and Zhuo Xu. 2023. Auto-tuning with reinforcement learning for permissioned blockchain systems. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1000–1012.
- [22] Pengze Li, Mingxuan Song, Mingzhe Xing, Zhen Xiao, Qiuyu Ding, Shengjie Guan, and Jieyi Long. 2024. Spring: Improving the throughput of sharding blockchain via deep reinforcement learning based state placement. In *Proceedings of the ACM Web Conference 2024*. 2836–2846.
- [23] Guoqiong Liao, Hao Ding, Chuanling Zhong, and Yinxiang Lei. 2024. Rt-dag: A dag-based blockchain supporting real-time transactions. *IEEE Internet of Things Journal* (2024).
- [24] Mingjin Liu, Fen R. Yu, Yuhang Teng, Victor C.M. Leung, and Min Song. 2019. Performance optimization for blockchain-enabled industrial internet of things (IIoT) systems: A deep reinforcement learning approach. *IEEE Transactions on Industrial Informatics* 15, 6 (2019), 3559–3570.
- [25] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. 2018. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems* 31 (2018).
- [26] Dexter Neo and Tsuhan Chen. 2023. DSAC-C: Constrained Maximum Entropy for Robust Discrete Soft-Actor Critic. *arXiv preprint arXiv:2310.17173* (2023).
- [27] Junpei Ni, Jiang Xiao, Shijie Zhang, Bo Li, Baochun Li, and Hai Jin. 2023. FLUID: Towards efficient continuous transaction processing in DAG-based blockchains. *IEEE Transactions on Knowledge and Data Engineering* 35, 12 (2023), 12679–12692.
- [28] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. 2021. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)* 54, 5 (2021), 1–35.
- [29] Xidi Qu, Shengling Wang, Kun Li, Jianhui Huang, and Xiuzhen Cheng. 2024. TidyBlock: A Novel Consensus Mechanism for DAG-based Blockchain in IoT. *IEEE Transactions on Mobile Computing* (2024).
- [30] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.
- [31] Mudita Sharma, Alexandros Komninos, Manuel López-Ibáñez, and Dimitar Kazakov. 2019. Deep reinforcement learning based parameter control in differential evolution. In *Proceedings of the genetic and evolutionary computation conference*. 709–717.
- [32] Meng Shen, Zhehui Tan, Dusit Niyato, Yuzhi Liu, Jiawen Kang, Zehui Xiong, Liehuang Zhu, Wei Wang, and Xuemin Shen. 2024. Artificial intelligence for web 3.0: A comprehensive survey. *Comput. Surveys* 56, 10 (2024), 1–39.
- [33] Jie Shi, Hong Wu, Dawei Luo, Haonan Gao, and Wei Zhang. 2023. Instantchain: Enhancing order-execute blockchain systems for latency-sensitive applications. In *International Conference on Database Systems for Advanced Applications*. Springer, 483–498.
- [34] Wellington Fernandes Silvano and Roderval Marcelino. 2020. Iota Tangle: A cryptocurrency to communicate Internet-of-Things data. *Future generation computer systems* 112 (2020), 307–319.
- [35] Yonatan Sompolsky, Yoav Lewenberg, and Aviv Zohar. 2016. Spectre: A fast and scalable cryptocurrency protocol. *Cryptology ePrint Archive* (2016).
- [36] Yonatan Sompolsky, Shai Wyborski, and Aviv Zohar. 2021. PHANTOM GHOSTDAG: a scalable generalization of Nakamoto consensus: September 2, 2021. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. 57–70.
- [37] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM international conference on management of data*. 1009–1024.
- [38] A Helen Victoria and Ganesh Maragatham. 2021. Automatic tuning of hyperparameters using Bayesian optimization. *Evolving Systems* 12, 1 (2021), 217–223.
- [39] Qin Wang, Jiangshan Yu, Shipping Chen, and Yang Xiang. 2023. SoK: DAG-based blockchain systems. *Comput. Surveys* 55, 12 (2023), 1–38.
- [40] Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, Bin Dai, and Qiguang Miao. 2022. Deep reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems* 35, 4 (2022), 5064–5078.
- [41] Chengru Wu, Beomseok Mehta, Mohammad Javad Amiri, Ryan Marcus, and Boon Thau Loo. 2022. Adachain: A learned adaptive blockchain. *arXiv preprint arXiv:2211.01580* (2022).
- [42] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. 2019. Hyperparameter optimization for machine learning models based on Bayesian optimization. *Journal of Electronic Science and Technology* 17, 1 (2019), 26–40.
- [43] Jie Xu, Yingying Cheng, Cong Wang, and Xiaohua Jia. 2021. Occam: A secure and adaptive scaling scheme for permissionless blockchain. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 618–628.
- [44] Ruoxi Xu. 2025. Fine Tuning for Dqn: Exploring Parameter Impacts on Cart Pole Performance. In *ITM Web of Conferences*, Vol. 78. EDP Sciences, 01006.
- [45] Haifeng Yu, Ivica Nikolic, Ruomu Hou, and Prateek Saxena. 2020. Ohie: Blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 90–105.
- [46] Jianting Zhang, Zicong Hong, Xiaoyu Qiu, Yufeng Zhan, Song Guo, and Wuhui Chen. 2020. Skychain: A deep reinforcement learning-empowered dynamic blockchain sharding system. In *Proceedings of the 49th International Conference on Parallel Processing*. 1–11.
- [47] Shijie Zhang, Jiang Xiao, Enping Wu, Feng Cheng, Bo Li, Wei Wang, and Hai Jin. 2024. Morphdag: A workload-aware elastic dag-based blockchain. *IEEE Transactions on Knowledge and Data Engineering* 36, 10 (2024), 5249–5264.
- [48] Xinyi Zhang, Hong Wu, Zhekang Chang, Shouwei Jin, Jiachen Tan, Fuliang Li, Tingting Zhang, and Bo Cui. 2021. Restune: Resource oriented tuning boosted by meta-learning for cloud databases. In *Proceedings of the 2021 International Conference on Management of Data*. 2102–2114.
- [49] Yue Zhang, Jian Lin, Zhipeng Lu, Qinghua Duan, and Shyh-Ching Huang. 2024. Pbl-tchain: A performance-enhanced permissioned blockchain for time-critical applications based on reinforcement learning. *Future Generation Computer Systems* 154 (2024), 301–313.

A Performance Calculation

For a given measurement window with k successfully confirmed transactions, we use the average latency:

$$\text{Latency}_{\text{avg}} = \frac{1}{k} \sum_{i=1}^k \text{Latency}_i$$

Throughput is defined as the number of confirmed transactions per unit of time, calculated between the first transaction's submission (T_{fs}) and the last transaction's confirmation (T_{lc}):

$$\text{Throughput} = \frac{k}{T_{lc} - T_{fs}}$$

B SAC Algorithm

Algorithm 1 SAC Algorithm

Require: Hyperparameters: $\gamma, \tau, \alpha_\pi, \alpha_Q, \alpha_\alpha$

- 1: Initialize networks: $\pi_\theta, Q_{\phi_1}, Q_{\phi_2}, Q_{\phi'_1}, Q_{\phi'_2}$, temperature α
 - 2: Initialize replay buffer \mathcal{D}
 - 3: **repeat**
 - 4: Collect trajectory using π_θ and store in \mathcal{D}
 - 5: Sample batch $\mathcal{B} = \{(s, a, r, s')\}$ from \mathcal{D}
 - 6: **Critic Update:**
 - 7: $y = r + \gamma(\min_j Q_{\phi'_j}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s'))$
 - 8: $\phi_j \leftarrow \arg \min_{\phi_j} \mathbb{E}_{(s,a) \sim \mathcal{B}} [(Q_{\phi_j}(s, a) - y)^2]$
 - 9: **Actor Update:**
 - 10: $\theta \leftarrow \arg \min_\theta \mathbb{E}_{s \sim \mathcal{B}} [\alpha \log \pi_\theta(\tilde{a}|s) - \min_j Q_{\phi_j}(s, \tilde{a})]$
 - 11: **Temperature Update:**
 - 12: $\alpha \leftarrow \arg \min_\alpha \mathbb{E}_{s \sim \mathcal{B}} [-\alpha(\log \pi_\theta(\tilde{a}|s) + \bar{H})]$
 - 13: **Target Update:**
 - 14: $\phi'_j \leftarrow \tau \phi_j + (1 - \tau) \phi'_j$
 - 15: **until** convergence
-

We employ the Soft Actor-Critic (SAC) algorithm, an off-policy method based on the maximum entropy reinforcement learning framework. SAC maintains five neural networks: a policy network π_θ , two critic networks Q_{ϕ_1} and Q_{ϕ_2} , and their corresponding target networks $Q_{\phi'_1}$ and $Q_{\phi'_2}$. During training, the agent collects trajectories using the current policy and stores transitions in a replay buffer \mathcal{D} . At each iteration, we sample a mini-batch \mathcal{B} and perform three updates. First, the critic networks minimize the Bellman error: $J_Q(\phi_j) = \mathbb{E}[(Q_{\phi_j}(s, a) - y)^2]$, where $y = r + \gamma(\min_j Q_{\phi'_j}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s'))$. Second, the policy network is updated to maximize $\mathbb{E}[\min_j Q_{\phi_j}(s, \tilde{a}) - \alpha \log \pi_\theta(\tilde{a}|s)]$ using the reparameterization trick. Third, the temperature parameter α is adjusted to maintain the target entropy. Finally, target networks are soft-updated with $\phi'_j \leftarrow \tau \phi_j + (1 - \tau) \phi'_j$.

C Hyperparameter Configuration for Alzo

Table 3: Hyperparameter Configuration for Alzo

Category	Parameter	Value
Constant	n	50
	k	10
High-Level Reward	λ_{thpt}	1.0
	λ_{lat}	0.5
	λ_{smooth}	0.1
Low-Level Reward	α	0.4
	β	0.4
	λ'_{smooth}	0.05
	η	2.0

In our implementation, both the policy network and value network are implemented as Multi-Layer Perceptrons (MLPs), each consisting of two hidden layers with 256 neurons per layer. The Rectified Linear Unit (ReLU) is employed as the activation function for all hidden layers. The networks are optimized using the Adam optimizer with a learning rate of 3×10^{-4} .

D State Transitions

For each domain controller $d \in \{N, L, F\}$, the transition function $P_{l,d}(s_{l,d,t+1} | s_{l,d,t}, \mathbf{a}_{l,d,t}, \mathbf{a}_{h,t})$ captures how local parameter adjustments and high-level strategic guidance jointly shape the next state:

$$s_{l,d,t+1} = f_d(s_{l,d,t}, \mathbf{a}_{l,d,t}, \mathbf{a}_{h,t}, \xi_{d,t}), \quad (9)$$

where f_d denotes the domain-specific transition function implemented by the blockchain transaction processing engine, $\mathbf{a}_{l,d,t}$ represents the controller's parameter adjustments, $\mathbf{a}_{h,t}$ contains the Meta-Controller's domain weight $a_{d,t}$ influencing resource allocation, and $\xi_{d,t}$ captures environmental randomness such as stochastic transaction arrivals, network latency, and nondeterminism in hardware resource load conditions.

Each controller exhibits unique transition characteristics: The Network Controller's state $\mathbf{s}_{l,N,t+1} = [B_{u,t+1}, Q_{\text{len},t+1}, a_{N,t+1}]$ evolves based on buffer size adjustments and random transaction arrival patterns. The Ledger Controller's state $\mathbf{s}_{l,L,t+1} = [T_{\text{tip},t+1}, a_{L,t+1}, O_{\text{rate},t+1}, C_{\text{usage},t+1}]$ transitions through consensus parameter changes and stochastic block propagation dynamics. The Full-Node Controller's state $\mathbf{s}_{l,F,t+1} = [a_{F,t+1}, Th_{n,t+1}, M_{\text{usage},t+1}]$ changes via cache adjustments and random state access patterns.