

ShardBenchmark: LLM-Driven Address Placement and Account Migration in Sharding Blockchains

Anonymous authors

Abstract

Sharding scales blockchains by parallelizing transactions across shards, but performance is limited by cross-shard interactions and load imbalance. Prior work on Address Placement (AP) and Account Migration (AM) often treats them separately and struggles with multi-objective constraints and long-horizon effects. We present ShardBenchmark, a fast-verifiable benchmark and simulator that unifies AP, AM, and their joint scheduling as constrained sequential decision-making tasks. Drawing on Ethereum statistics, we build controllable synthetic datasets that capture head-heavy activity, temporal drift, and community clustering, and evaluate policies with cross-shard ratio, load balance, and throughput metrics. We further propose an LLM-driven decision framework that compresses observations into prompts and outputs structured actions with auditable rationales. Experiments across various LLMs show that stronger models consistently outperform a Random baseline, with scale and reasoning quality proving critical—while smaller models may generate infeasible plans. ShardBenchmark offers the first unified, reproducible testbed for evaluating LLM-based schedulers in sharding blockchains. Code and datasets will be released upon paper acceptance.

CCS Concepts

- Theory of computation → Algorithmic mechanism design;
- Computer systems organization → Peer-to-peer architectures.

Keywords

sharding, LLMs, address placement, account migration

ACM Reference Format:

Anonymous authors. 2025. ShardBenchmark: LLM-Driven Address Placement and Account Migration in Sharding Blockchains. In *Proceedings of Proceedings of the ACM Web Conference 2026 (WWW '26)*. ACM, New York, NY, USA, 11 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

Sharding technology has emerged as an important pathway for scaling blockchains [26]. By partitioning transaction data and account state across multiple parallel shards, each responsible for a subset of the workload, sharding significantly boosts throughput and mitigates congestion without undermining decentralization [23]. These

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WWW '26, April 13-17, 2026, Dubai, United Arab Emirates

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
<https://doi.org/XXXXXXX.XXXXXXX>

capabilities are critical for supporting the demands of large-scale decentralized applications [5, 21].

Despite its promise, sharding introduces two persistent bottlenecks in engineering practice: (i) **frequent cross-shard interactions** [11, 33], which amplify communication and scheduling overhead, and (ii) **load imbalance across shards** [10, 18], which undermines resource utilization and stability. Two control levers are widely recognized as central to addressing these issues: **Address Placement (AP)** [19], which decides where to place newly created accounts upon their first appearance; and **Account Migration (AM)** [28], which periodically reallocates existing accounts across shards to reshape load distributions. These tasks are inherently combinatorial and sequential, with placement decisions influencing future migrations by determining which shards should handle specific accounts, and migration outcomes reshaping subsequent placements by redistributing accounts based on load balance and shard performance [10, 15].

Traditional heuristics and reinforcement learning (RL) methods have shown partial improvements but struggle under the practical sharding blockchain regime of multi-objective optimization, strict feasibility constraints, and long-horizon sequential dependencies [28]. Recent advances in Large Language Models (LLMs) suggest an alternative modeling paradigm [24]. LLMs can compress high-dimensional observations into textual prompts, reason over multiple objectives, and output executable structured actions under explicit constraints [36]. Moreover, LLMs offer auditable “plan-to-action” rationales, making them attractive for complex blockchain scheduling tasks where interpretability and adaptability are as critical as raw efficiency [6]. In practice, this means LLMs can provide decisions that are simultaneously traceable and adaptable to new workloads, a property difficult to replicate with fixed heuristics or narrowly trained RL models [7, 30].

However, three major gaps remain. First, evaluations of AP and AM largely depend on noisy raw blockchain traces, where exogenous factors intricately intertwine with task-relevant variables, weakening causal interpretability [19, 28]. Second, the field lacks shard-specific benchmarks and fast-verifiable simulation pipelines to enable systematic comparisons, reproducibility, and controlled ablation [29]. Third, mapping LLM capabilities to concrete sharding tasks remains underexplored, particularly under unified formulations that combine both AP and AM. These gaps hinder both principled research and engineering deployment, and they highlight the necessity of constructing controlled synthetic datasets and standardized evaluation pipelines.

Empirical analyses of Ethereum reveal salient structural regularities that motivate our study: head-heavy activity distributions with temporal drift, strong community structures, and observable periodic hot-spot migration [11, 20, 35]. In particular, we find that a small fraction of addresses consistently generate over 70–80% of transactions, yet the set of dominant addresses changes over short

horizons. These dynamics directly shape scheduling: aggressively colocating new accounts with introducers may reduce cross-shard traffic but exacerbate shard imbalance; conversely, over-frequent migrations can temporarily depress throughput. Such trade-offs highlight the intrinsic coupling between AP and AM, calling for unified modeling and evaluation.

To address these challenges, we propose **ShardBenchmark**, a fast-verifiable benchmark and simulation environment for AP and AM in sharding blockchains. ShardBenchmark distills essential statistical laws from Ethereum traces and constructs controllable synthetic datasets that amplify head effects, temporal continuity, and community clustering. This yields both realistic and tunable workloads for robust algorithm evaluation. On top of ShardBenchmark, we formalize AP, AM, and their unified scheduling as constrained sequential decision-making tasks. We also design multi-metric evaluation protocols for cross-shard ratio, load balance, and throughput and develop an LLM-driven reference policy framework that maps compressed observations into constrained actions. Experimental comparisons across various LLMs, such as DeepSeek-R1[8], GPT-4.1[25], Claude[2], Gemini[4], and Qwen3[32] variants, demonstrate the feasibility of this approach, revealing both the strengths and limitations of LLM agents in real-chain settings. In addition, following prior work [19, 28], we implement a PPO [27] algorithm and a Random algorithm as controls to provide references for performance comparison.

In summary, our contributions are:

- **ShardBenchmark.** We present a unique benchmark that unifies AP, AM, and their joint coordination AP+AM, providing fast-verifiable, and reproducible evaluation pipelines.
- **LLM-based decision framework.** We propose an innovative LLM-driven scheduler that encodes shard states into permutation-invariant prompts and outputs constrained, interpretable actions for AP and AM.
- **Comprehensive results and insights.** Experiments show that strong LLMs (e.g., DeepSeek-R1, GPT-4.1) significantly outperform Random and heuristic baselines, demonstrating the feasibility and potential of LLMs as effective schedulers for sharding blockchains, and highlighting the benefits of unified AP+AM coordination.

2 Task Modeling

The ShardBenchmark overall workflow is shown in Figure 1. For clarity, all notations are systematically summarized in Appendix A. Our goal is to capture the fundamental control levers that govern performance in sharding blockchains, namely AP, AM, and their joint coordination AP+AM. We therefore design three sequential decision-making tasks that together form the backbone of our benchmark. The first is AP, where at every block the system must decide how to assign newly observed addresses to shards. The second task is AM, which enables the system to periodically reallocate existing accounts across shards to alleviate imbalances and reduce cross-shard traffic.

In practice, placement and migration are not completely independent levers but tightly coupled mechanisms: every block requires AP, while migration opportunities arise periodically and directly interact with placement outcomes. To capture this interdependence,

we introduce a unified task AP+AM that combines AP and AM. The procedure of the three tasks is summarized in Appendix C.

2.1 Address Placement (AP)

AP aims to assign newly observed account addresses to appropriate shards on a per-block, streaming basis so as to simultaneously reduce cross-shard communication overhead and improve load balancing. For every processed block, the system must place the addresses that appear for the first time, thereby affecting subsequent routing patterns and global performance.

AP is inherently a constrained sequential decision-making problem. Blocks arrive over time, and at each time step t the agent chooses placements for the new addresses based on currently observable information. These decisions immediately alter the statistics and efficiency of the current block as well as subsequent blocks, and in combination with time-varying traffic spectra and evolving community structures, they induce a non-stationary sequence. It is therefore natural and necessary to formalize AP as a Markov Decision Process (MDP) [19]. Formally, we represent it as a tuple $\mathcal{M}_{\text{AP}} = (\mathcal{S}^1, \mathcal{A}^1, R, \mathcal{P}^1, \gamma)$ where \mathcal{S} denotes the state space, \mathcal{A} the action space, R the reward function, and \mathcal{P} the transition dynamics. γ is the discounted factor, $0 < \gamma \leq 1$.

State Space \mathcal{S}^1 . We define the state at block index t as:

$$s_t^1 = \{t, \mathbf{a}, \mathbf{c}, \ell, V, M, \mathbf{s}\}. \quad (1)$$

Here, $\mathbf{a} \in \mathbb{Z}^K$ denotes the aggregate vector of total transactions per shard, and $\mathbf{c} \in \mathbb{Z}^K$ denotes the vector of cross-shard transactions. The set $\ell = [a_1, \dots, a_N]$ contains the newly observed addresses in the current block, where $N = |\ell|$ represents the total number of newly observed addresses. For each new address a_j , the row \mathbf{v}_j of the source matrix $V \in \mathbb{Z}^{N \times K}$ records its interaction counts with each shard, and the mapping $M = \{a_j \mapsto \mathbf{v}_j\}_{j=1}^N$ associates each address with its source vector. Finally, $\mathbf{s} \in \mathbb{Z}^K$ tracks the current number of accounts stored in each shard. All variables are non-negative integers.

Action Space \mathcal{A}^1 . At each time step t , the agent outputs an assignment dictionary for all new addresses

$$a_t^1 : \ell \rightarrow \{0, 1, \dots, K-1\}, \quad \text{dom}(a_t) = \ell, \quad (2)$$

i.e., for each $a_j \in \ell$ it specifies a target shard $k \in \{0, \dots, K-1\}$. The action must cover all new addresses; no omissions or out-of-range targets are allowed.

Reward Function \mathcal{R} . The one-step reward at the t -th block index multiplicatively combines measured cross-shard efficiency and overall load balancing.

$$R_t = r_t^\alpha b_t^\beta, \quad \alpha, \beta > 0, \quad (3)$$

$$r_t = \left(1 - 0.5 \cdot \frac{\sum_{k=1}^K c_k}{\sum_{k=1}^K a_k + \varepsilon}\right)^{\gamma_r}, \quad (4)$$

$$b_t = \left(1 + \exp\left(k \left(\frac{\sigma(\mathbf{a})}{\mu(\mathbf{a}) + \varepsilon} - \tau\right)\right)\right)^{-1}. \quad (5)$$

Here $\gamma_r > 0$ is a tunable exponent that controls the nonlinearity of cross-shard penalization, $k > 0$ and $\tau \in (0, 1)$ are hyperparameters for the balancing term, μ and σ denote the mean and standard deviation, respectively, and $\varepsilon > 0$ is a numerical stabilizer. This

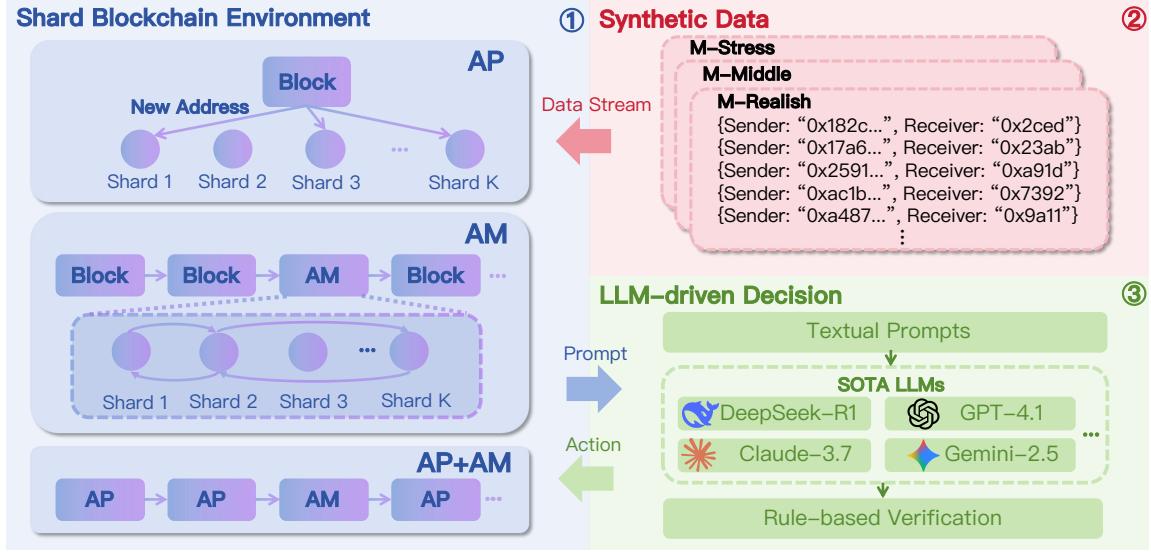


Figure 1: The ShardBenchmark overall workflow. Synthetic datasets provide controlled transaction streams, while LLMs generate and verify shard actions, both interacting with the shard blockchain environment.

implementation is inspired by the approach used in the sharding blockchain framework block-emulator [12].

State Transition \mathcal{P}^1 . The system proceeds block by block in a streaming manner: load the next block (a set of sender-receiver pairs), compute \mathbf{a}, \mathbf{c} under the current address-to-shard mapping, identify new addresses and construct (ℓ, V, M) , and form s_{t+1} . Placement decisions take effect within the current step and influence subsequent statistics and rewards.

2.2 Account Migration (AM)

AM addresses the reallocation of existing accounts across shards in order to reduce cross-shard transaction ratios and improve load balancing. The system triggers migration decisions periodically: migration operations are permitted every P steps, specifically at steps where $(i + 1) \bmod P = 0$, where P represents the migration period. At these steps, the agent is allowed to perform up to a migration operations; at all other steps, no migration is applied (no-op). Similar to AP, AM is naturally a constrained sequential decision-making problem: migration actions taken at specific intervals reshape the shard distribution of existing accounts, directly influencing both the current block statistics and the efficiency of subsequent blocks. This process is therefore also formalized as an MDP $\mathcal{M}_{AM} = (\mathcal{S}^2, \mathcal{A}^2, R, \mathcal{P}^2, \gamma)$.

State Space \mathcal{S}^2 . We define the state at block index t as:

$$s_t^2 = \{ t, \mathbf{a}, \mathbf{c}, \mathbf{s}, O \}. \quad (6)$$

Here, O is a snapshot of prefix-shard statistics from the previous block: for each shard $u \in \{0, \dots, K-1\}$ and each hexadecimal prefix $p \in \{0, \dots, f\}$, we store a vector $\mathbf{u}_{u,p} \in \mathbb{Z}^{K+1}$, where $\mathbf{u}_{u,p}[q]$ records the number of interactions with target shard q ($q = 0$ to $(K - 1)$), and $\mathbf{u}_{u,p}[K]$ stores the total count in the previous block.

Action Space \mathcal{A}^2 . At each migration step, i.e., when $(i+1) \bmod P = 0$, the agent can execute up to a migrations:

$$a_t^2 = [(u_1 \rightarrow v_1, p_1), \dots, (u_m \rightarrow v_m, p_m)], \quad m \leq a, \quad (7)$$

where $u_j \neq v_j \in \{0, \dots, K-1\}$ and $p_j \in \{0, \dots, f\}$. Each tuple $(u_j \rightarrow v_j, p_j)$ denotes migrating all addresses with first-byte prefix p_j from shard u_j to shard v_j . At non-migration steps, $a_t^2 = \emptyset$.

Reward Function \mathcal{R} . The reward function is defined consistently with AP, using post-migration statistics at the t -th block $R_t = r_t^\alpha b_t^\beta$.

State Transition \mathcal{P}^2 . At migration steps t , the chosen action a_t updates the address-to-shard mapping ϕ and shard account sets $\{\mathcal{S}_t^2\}$. Subsequently, the per-block statistics (\mathbf{a}, \mathbf{c}) and reward R_t are recomputed under the updated mapping. The prefix-shard snapshot O is refreshed using the current block, and the next block is loaded to form s_{t+1}^2 .

2.3 AP+AM Unified Scheduling

While AP and AM can be studied in isolation, in practice they jointly determine the efficiency of a sharding blockchain. In every block, the system must place new addresses (AP), while periodically—every P blocks—it can also perform up to a prefix-based AM. The unified environment thus integrates both mechanisms: each time step consists of placing new addresses, optionally migrating old addresses, and then recomputing shard-level statistics and rewards. The observation space $\mathcal{S}^3 = \mathcal{S}^1 \cup \mathcal{S}^2$ is the union of the AP and AM elements. Rewards and evaluation metrics remain identical, combining cross-shard efficiency and load balancing. This unified formulation provides a realistic environment in which sequential coordination between AP and AM can be directly evaluated.

3 Synthetic Data Generation

Given the inherent complexity of AP and AM in sharding blockchains, as outlined in Sections 2, it is crucial to establish a robust synthetic data generation pipeline. This pipeline will not only enable systematic evaluation of various AP and AM algorithms but also address the challenging nature of creating realistic test environments. Such data is essential for effectively assessing the performance and

scalability of these algorithms in realistic, large-scale blockchain scenarios. This pipeline emulates transaction patterns observed in real blockchain environments while allowing key parameters to be controlled for testing algorithmic robustness under diverse scenarios. The generation model is grounded in empirical observations of Ethereum data and is instantiated through three representative datasets (**M-Realish**, **M-Middle**, and **M-Stress**). Below we describe the generative mechanisms in detail.

Sender–Receiver Model. We assume K shards, G communities, and N_{old} pre-existing addresses. Each address a is associated with a community $g(a) \in \{1, \dots, G\}$ and a shard $s(a) \in \{0, \dots, K-1\}$. The sender distribution exhibits a head effect and hotspot amplification. Let $w(a)$ denote the base activity weight of address a . At each block, we sample a hotspot shard $h \in \{0, \dots, K-1\}$, and apply an amplification factor γ_a so that the normalized probability $\tilde{w}(a)$ of selecting a as sender is

$$\tilde{w}(a) = \frac{w(a) \cdot (1 + (\gamma_a - 1) \cdot \mathbf{1}[s(a) = h])}{\sum_{x \in \mathcal{A}_{\text{old}}} w(x) \cdot (1 + (\gamma_x - 1) \cdot \mathbf{1}[s(x) = h])}, \quad (8)$$

The receiver distribution combines three mechanisms: reuse of previously introduced new addresses, community-driven sampling with an intra/inter ratio, and sporadic introduction of fresh addresses. For each sender s , we maintain a reuse budget B_s . With probability $\eta \approx p_{\text{repeat_new}}$, the receiver is uniformly chosen from the set \mathcal{N}_s of new addresses historically introduced by s . Otherwise, the receiver is drawn from a mixture $\Pr(R = r \mid S = s)$:

$$\Pr(R = r \mid S = s) = p_{\text{new}} \cdot \delta(r \notin C) + (1 - p_{\text{new}}) \cdot p_{\text{old}}(r \mid S = s), \quad (9)$$

$$p_{\text{old}}(r \mid S = s) = p_{\text{intra}} \cdot \text{Unif}(C_{g(s)} \setminus \{s\}) + (1 - p_{\text{intra}}) \cdot \text{Unif}(C \setminus C_{g(s)}). \quad (10)$$

Here, p_{new} denotes the probability that the receiver is a newly introduced address, and $\delta(r \notin C)$ is an indicator ensuring the receiver is drawn from the set of unseen addresses, where C denotes the set of all previously observed addresses. The term $p_{\text{old}}(r \mid S = s)$ corresponds to the case where the receiver is an existing address. Within this, p_{intra} specifies the probability of sampling an address from the same community as the sender, while $(1 - p_{\text{intra}})$ covers cross-community selections. Finally, $C_{g(s)}$ denotes the set of observed addresses in the community $g(s)$ of sender s , and C denotes the set of all observed addresses. This mixture captures three fundamental behaviors of blockchain transactions: the introduction of new accounts, the cohesion of community-driven activity, and the presence of cross-community interactions.

Sticky Placement of New Addresses. When a new address r is introduced as receiver, its shard is assigned according to a receiver-first stickiness rule:

$$s(r) = \begin{cases} s(S), & \text{with probability } p_s, \\ \text{Unif}(\{0, \dots, K-1\} \setminus \{s(S)\}), & \text{with probability } 1 - p_s. \end{cases} \quad (11)$$

Thus new addresses tend to colocate with their introducer's shard, while allowing controlled spillover across shards.

Head Effect Calibration. To reproduce the heavy-tailed activity distribution observed in real data, the sender weights $w(a)$ are modeled as

$$w(a) \sim \text{Lognormal}(\mu, \sigma^2). \quad (12)$$

Given a target head-coverage constraint—namely that the smallest p_t of addresses cover approximately 80% of the total transaction volume—we tune σ via binary search to satisfy

$$\frac{\sum_{i=1}^k w(i)}{\sum_{j=1}^{N_{\text{old}}} w_j} \approx 0.8, \quad k = \lceil \text{top_pct} \cdot N_{\text{old}} \rceil, \quad (13)$$

where $w_{(1)} \geq w_{(2)} \geq \dots$ denotes the descending order of weights. This procedure achieves calibration within a numerical tolerance of $\pm 2\%$. The three synthetic datasets differ in their stickiness p_s , head-effect concentration p_t , and reuse probability p_r , as summarized in Table 1.

Table 1: Parameters of the three synthetic datasets

Dataset	p_s	p_t	p_r
M-Realish	0.20	0.10	0.20
M-Middle	0.40	0.20	0.40
M-Stress	0.60	0.40	0.60

The three varying parameters capture complementary aspects of transaction dynamics. The stickiness parameter p_s controls how likely a newly introduced address collocates with its introducer's shard, directly shaping the placement of fresh accounts and thus serving as the key driver for AP. The head-effect parameter p_t controls the concentration of transaction volume among the most active addresses, which determines whether load is dominated by a small subset of accounts and therefore highlights the importance of AM in redistributing them. Finally, the reuse probability $p_{\text{repeat_new}}$ regulates how often newly created addresses reappear in later transactions, linking short-term placement choices to long-term workload persistence. By varying these three parameters across datasets, we obtain environments that stress different control levers—AP sensitivity to new-address stickiness and AM sensitivity to head-effect concentration—thereby enabling balanced evaluation of both AM and AP mechanisms.

4 LLM-driven Decision

We unify AP and AM into a structured, constrained sequential decision-making problem. At each time step t , the agent observes the system state s_t and produces a joint action

$$a_t = (a_t^{\text{AP}}, a_t^{\text{AM}}). \quad (14)$$

Unlike traditional heuristics, we treat the LLM as an implicit optimizer: the key statistics are compressed into compact textual prompts, and the LLM outputs an approximately optimal decision under explicit action parameterization and feasibility constraints. The overall objective is to maximize the discounted cumulative reward

$$\max_{\{a_t\}_{t=0}^{T-1}} \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t), \quad R(s_t, a_t) = r_t^\alpha b_t^\beta, \quad (15)$$

where r_t denotes cross-shard efficiency, b_t denotes load balancing, and $\alpha, \beta > 0$ are weighting hyperparameters.

Observation Compression and Prompt Construction. The raw environment state contains high-dimensional statistics such as the new-address source matrix, per-shard transaction vectors, and prefix-shard snapshots from the previous block. To enable the LLM to “read” and reason effectively, a deterministic serialization operator $\mathcal{T}(\cdot)$ is used to generate textual prompts:

$$x_t^{\text{AP}} = \mathcal{T}(\text{AP}(s_t)), \quad x_t^{\text{AM}} = \mathcal{T}(\text{AM}(s_t)). \quad (16)$$

The design principle is sufficiency and symmetry: retaining statistics monotonically related to R_t (e.g., cross-shard ratio and coefficient of variation), while ensuring permutation-invariant representations with respect to shard relabeling. A prompt example x_t^{AP} is shown in Appendix D.

Action Parameterization and Feasible Domain. To convert the combinatorial action space into a compact structure suitable for LLM outputs, we adopt two parameterizations:

(i) AP. The LLM outputs a proportion vector $\mathbf{p}_t \in [0, 1]^K$, interpreted via a deterministic mapping G_{AP} :

$$a_t^{\text{AP}} = G_{\text{AP}}(s_t, \mathbf{p}_t), \quad \mathbf{p}_t = f_{\theta}^{\text{AP}}(x_t^{\text{AP}}), \quad (17)$$

where f_{θ}^{AP} denotes the LLM-driven mapping. Here, the output vector \mathbf{p}_t specifies, for each shard i , the fraction p_t^i of new addresses from senders on shard i that remain in the same shard to reduce cross-shard transactions, while the remaining $1 - p_t^i$ are reassigned to maintain load balance.

(ii) AM. The LLM outputs a set of at most m migration pairs

$$M_t \subseteq \{0, \dots, K-1\}^2, \quad |M_t| \leq m, \quad (18)$$

with the constraint that for each $(u, v) \in M_t$, $u \neq v$. For each pair, a prefix $p \in \{0, \dots, f\}$ is chosen using the prefix-shard statistics from the previous block. The final migration decision is

$$a_t^{\text{AM}} = G_{\text{AM}}(s_t, M_t), \quad M_t = f_{\theta}^{\text{AM}}(x_t^{\text{AM}}). \quad (19)$$

Here, G_{AM} selects, for each migration pair (u, v) , the prefix with the highest number of cross-shard transactions between the two shards and exchanges the corresponding accounts accordingly.

The feasible domain is thus

$$C(s_t) = \left\{ (a^{\text{AP}}, a^{\text{AM}}) \mid \text{dom}(a^{\text{AP}}) = \ell_t, |a^{\text{AM}}| \leq m \right\}, \quad (20)$$

where ℓ_t is the set of new addresses in the current block, and AM only enabled at periodic steps.

LLM as an Implicit Optimizer. The LLM output can be viewed as an approximate solution to

$$\hat{a}_t = \arg \max_{a \in C(s_t)} \widehat{\Delta R}(s_t, a), \quad (21)$$

where $\widehat{\Delta R}$ denotes a proxy objective encoded in the prompt (e.g., “reduce cross-shard ratio and variance” with explicit weighting). The prompt specifies objectives, constraints, and output format in natural language, thereby transforming a combinatorial optimization problem into a constrained text-to-structure mapping. To enhance interpretability, the LLM also produces a short textual rationale, although the environment only consumes the structured action. This procedure is summarized in Algorithm 1.

Algorithm 1: LLM-driven Decision for AP & AM

```

Input: Number of shards  $K$ , migration period  $P$ ,  

       hyperparameters  $\alpha, \beta, k, \tau, \gamma_r$ ; address map  $M$ ,  

       transaction block stream; LLM backend  

Output: Historical logs  $\mathcal{L}$  and aggregated metrics  

1 Initialize: Load the first block  $s_0$ ; initialize address mapping  

   and logs  

2 while data not exhausted do  

   // AP decision  

3   Construct AP prompt ( $\mathbf{a}, \mathbf{c}, V$ ) and query LLM  

   →  $\mathbf{p}_t \in [0, 1]^K$   

4   Apply  $G_{\text{AP}}$  to assign new addresses according to  $\mathbf{p}_t$   

   // AM decision (only at periodic steps)  

5   if  $(i+1) \bmod P = 0$  then  

6     Construct AM prompt ( $\mathbf{a}, \mathbf{c}, O$ ) and query LLM  

     →  $\{(u \rightarrow v)\}_{\leq m}$   

7     For each  $(u \rightarrow v)$ , select prefix  $p$  from  $M[u][p][v]$   

       and migrate accounts  

   // Reward update  

8   Recompute  $\mathbf{a}, \mathbf{c}$  under updated mapping  

9   Compute  $R_t = r_t^\alpha b_t^\beta$  and log  $\{i, |\ell|, \text{migrated}, r_t, b_t, R_t\}$   

10  Update  $M$  from the current block and load the next  

    block  $s_{t+1}$   

11 return  $\mathcal{L}$  and aggregated metrics

```

5 Experiments

We conducted experiments to evaluate the performance of different models across three sequential decision-making tasks: AP, AM, and their unified scheduling (AP+AM). Three datasets—M-Realish, M-Middle, and M-Stress—are used, varying in stickiness p_s , head-effect concentration p_t , and reuse probability p_r . These variations stress different aspects of AP and address migration (AM), allowing for a comprehensive evaluation of algorithm performance under diverse conditions. Specifically, M-Realish mimics Ethereum-like statistics, M-Middle provides moderate skew, and M-Stress represents highly adversarial workload conditions. We benchmark a diverse set of large language models (LLMs), including DeepSeek-R1, GPT-4.1, Claude, Gemini, and multiple variants of Qwen3 (8B, 13B, 32B). For each model, we query with consistent prompt templates and enforce structured outputs. As a baseline, we include Random allocation, which provides the minimum performance reference and allows us to quantify the relative improvement achieved by each LLM. We note that the Qwen3-4B model often produced illogical rationales that failed to translate into valid structured actions, and is therefore excluded from the reported results. The hyperparameter settings used in ShardBenchmark are summarized in Appendix B.

5.1 Task Results

The results for AP across the three datasets are summarized in Table 2. Across all three datasets, most LLMs outperform the Random baseline, confirming that language models can exploit structure in sender-receiver statistics to reduce cross-shard traffic while keeping shard loads reasonable. DeepSeek-R1 leads consistently, with GPT-4.1 and Claude close behind and also improving as the scenario

becomes more skewed. Gemini is above Random in every dataset but trails the top models. A notable exception is Qwen3-8B, which underperforms Random on AP across all datasets, suggesting that small-capacity models struggle to translate compressed observations into effective placement proportions. A second clear trend is that scores rise from M-Realish to M-Stress for the stronger models. This is consistent with the data generator: higher stickiness p_s and stronger head concentration p_t make new-account interactions more predictable (more sender-receiver colocations), so models that recognize these regularities can cut cross-shard edges; the benefit on r_t (cross-shard term) often outweighs the balancing penalty b_t , yielding higher overall reward. We also include a Least-Loaded (LL) heuristic, which always assigns new addresses to the shard with the smallest current load. While LL yields slight improvements over Random, its overall effect remains limited.

The results for AM are shown in Table 3. In AM task, absolute rewards are lower than AP across the board because migration is (i) periodic (only every P blocks) and (ii) constrained (prefix-group moves), so its immediate effect is smaller. Still, DeepSeek-R1 and GPT-4.1 consistently beat Random and improve as workloads become more skewed, indicating they capture useful long-horizon rebalancing signals. Claude and Gemini are close to Random on Realish/Middle but edge ahead on Stress, while Qwen3-8B/13B lag or match Random, reflecting difficulty extracting robust decisions from prefix-shard tensors. The underlying cause is that AM’s efficacy depends on reasoning over global load distributions and temporal drift; periodic actions must anticipate future traffic rather than optimize the current block, which favors larger or stronger models with better long-horizon pattern use.

In the joint task AP+AM (Table 4), every strong LLM beats Random, and rewards are notably higher than AM alone. DeepSeek-R1 again tops all datasets, GPT-4.1 and Claude are competitive, with some non-monotonicity on Stress for Claude/Gemini). Qwen3 shows clear scale benefits: 8B underperforms Random, but 13B and especially 32B surpass Random with healthy margins. The trend that stronger models improve from Realish to Stress persists, echoing AP: as stickiness and head-concentration intensify, AP removes a larger share of cross-shard edges at birth, while AM periodically “cleans up” accumulation from head addresses and hotspot drift—a complementary effect that unified scheduling can exploit. Where some mid-tier models dip on Stress, such as Gemini and Claude, the likely cause is that tighter coupling between placement and migration makes feasibility/format errors or suboptimal trade-offs more costly; larger models better juggle the two levers and maintain gains on both r_t and b_t .

Across all three tasks, the PPO algorithm does not exhibit clear improvements over the Random baseline. We attribute this to the lack of a dedicated training phase: unlike LLMs, RL-based algorithm requires substantial interaction-driven optimization to discover effective policies, suggesting that future work should incorporate a well-designed training process.

5.2 Cross-Shard Efficiency & Load Balance

Figure 2 presents the evolution of the cross-shard transaction ratio when DeepSeek-R1 performs the AP+AM task on three synthetic datasets (M-Realish, M-Middle, and M-Stress). In all cases, the ratio

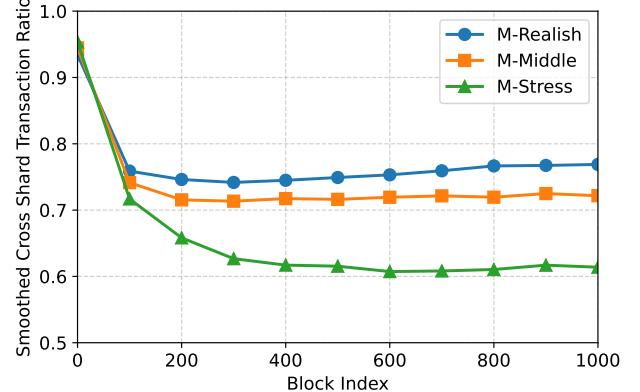


Figure 2: Cross-shard transaction ratio results.

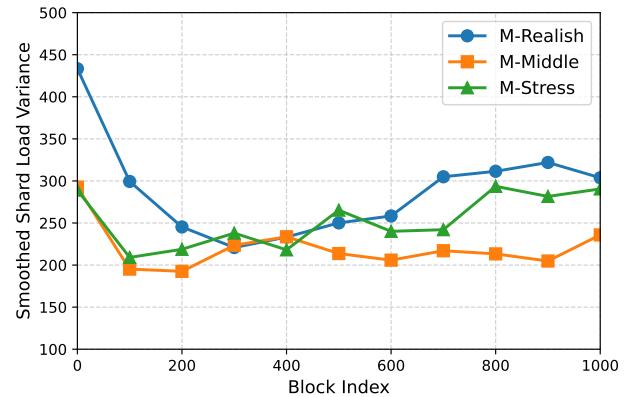


Figure 3: Shard load variance results.

drops rapidly from nearly 1.0 during the initial stage, indicating that early account placement and migration decisions can significantly reduce cross-shard overhead. The curves then enter a relatively stable phase, with clear differences across datasets. Notably, in M-Realish the ratio exhibits a slight upward trend in the later stage, suggesting room for further optimization under long-term execution. Overall, these results demonstrate that the difficulty and structural properties of the synthetic datasets directly affect the convergence level of cross-shard ratios, with the M-Stress environment most clearly highlighting the advantages of the scheduling strategy. This validates the benchmark’s ability to distinguish algorithmic performance under varying tasks and workload conditions.

Figure 3 presents the evolution of shard load variance when DeepSeek-R1 performs the AP+AM task on the three synthetic datasets. At the beginning, M-Realish shows a very high variance, which quickly drops as early account placement and migration decisions redistribute workloads across shards. M-Middle and M-Stress start at lower levels and stabilize faster. In the later stage, however, M-Realish exhibits a steady upward trend, and M-Stress also fluctuates with occasional spikes, suggesting that load imbalance may re-emerge under long-term dynamics. These results highlight that workload distribution stability is highly dependent on dataset characteristics: while the benchmarked strategy can effectively suppress load imbalance initially, long-horizon conditions such as hotspot drift and head concentration can reintroduce variance, underscoring the importance of adaptive scheduling mechanisms.

Table 2: Performance of AP on Different Datasets

Dataset	DeepSeek-R1	GPT-4.1	Claude	Gemini	Qwen3-8B	Qwen3-13B	Qwen3-32B	Random	LL	PPO
M-Realish	0.2146	<u>0.1856</u>	0.1810	0.1527	0.0781	0.1489	0.1540	0.1078	0.1273	0.1094
M-Middle	0.2184	<u>0.2081</u>	0.1980	0.1726	0.0828	0.1563	0.1678	0.1143	0.1304	0.1146
M-Stress	0.2681	<u>0.2390</u>	0.1972	0.1648	0.0877	0.1812	0.1837	0.1168	0.1422	0.1201

Table 3: Performance of AM on Different Datasets

Dataset	DeepSeek-R1	GPT-4.1	Claude	Gemini	Qwen3-8B	Qwen3-13B	Qwen3-32B	Random	PPO
M-Realish	<u>0.068</u>	0.069	0.064	0.055	0.039	0.045	0.059	0.062	0.064
M-Middle	0.083	<u>0.079</u>	0.074	0.066	0.045	0.049	0.055	0.075	0.072
M-Stress	0.088	<u>0.084</u>	0.082	0.080	0.052	0.058	0.077	0.078	0.080

Table 4: Performance of Combined AP and AM on Different Datasets

Dataset	DeepSeek-R1	GPT-4.1	Claude	Gemini	Qwen3-8B	Qwen3-13B	Qwen3-32B	Random	PPO
M-Realish	0.2496	<u>0.2218</u>	0.2139	0.1807	0.0921	0.1792	0.1832	0.1322	0.1319
M-Middle	0.2514	<u>0.2423</u>	0.2331	0.2076	0.0958	0.1828	0.1956	0.1316	0.1320
M-Stress	0.3085	<u>0.2842</u>	0.2276	0.1923	0.1011	0.2091	0.2125	0.1368	0.1338

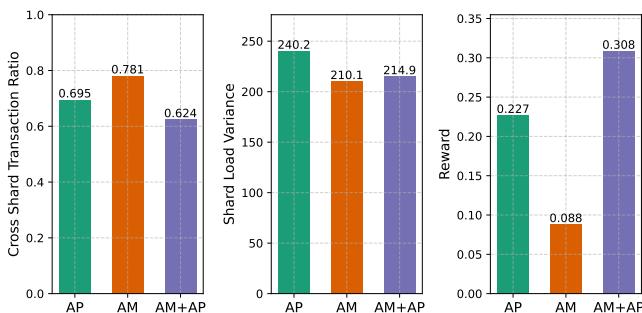
**Figure 4: Shard load variance results.**

Figure 4 summarizes the performance of DeepSeek-R1 across the three tasks (AP, AM, and AP+AM) on three key metrics: cross-shard transaction ratio, shard load variance, and overall reward. The results reveal clear trade-offs. First, AP alone substantially reduces the cross-shard transaction ratio compared to AM, while AM achieves the lowest shard load variance. Second, when the two mechanisms are combined (AP+AM), the model balances both objectives, achieving the lowest cross-shard ratio while keeping variance relatively low. Most importantly, the reward metric highlights a striking pattern: AM alone contributes very little, indicating that periodic migrations by themselves cannot effectively improve end-to-end performance. By contrast, AP provides a much higher reward, and AP+AM achieves the best overall outcome by coupling immediate placement efficiency with long-horizon rebalancing. These findings demonstrate that while AM is crucial for stabilizing load distributions, its standalone effect on reward is limited, and real gains emerge only when it is integrated with AP.

5.3 Case Study: DeepSeek-R1 on AP+AM

```
<reasoning>
To determine the optimal diversion proportions, I need to
balance two objectives...
First, analyze the current state:
- Calculate the cross-shard ratio ...
- Identify overloaded shards and those with high cross-shard
...
Let me calculate the cross-shard ratio and normalize the
load:
- Shard 0: 99/127 = 0.78 cross-shard ratio, medium load ...
- Shard 6: 106/134 = 0.79 cross-shard ratio, high load ...
Based on this analysis:
- Shards with high load (2, 6, 11, 15) should have lower p_i
values to divert traffic...
Balancing these factors, I'll set p_i values:
- For high-load shards: lower values (0.3-0.5)...
</reasoning>
<answer>[0.65, 0.6, 0.4, 0.65, 0.7, 0.65, 0.5, 0.65, 0.6,
0.8, 0.65, 0.35, 0.7, 0.65, 0.8, 0.3]</answer>
```

To further illustrate how large models coordinate placement and migration, we present a case study of DeepSeek-R1 on the AP+AM task under the M-Middle dataset. The full prompt and response are in Appendix D. DeepSeek-R1’s reasoning process demonstrates a clear trajectory: the model first decomposes the objective into two parts—reducing cross-shard transaction ratios and improving load balance—and then analyzes the statistics of each shard individually. Specifically, DeepSeek-R1 computes the cross-shard ratio for every shard and uses the total transaction volume to identify overloaded shards and those with high cross-shard ratios. Based on this analysis, it adjusts the allocation proportions p_i . Ultimately, the output reflects a balanced strategy: heavily loaded shards typically fall within the 0.3–0.5 range, lightly loaded shards within

0.7–0.9, and most medium-load shards remain in the 0.5–0.7 range. This trajectory, accompanied by a coherent rationale, shows that DeepSeek-R1 is not merely generating actions mechanically but is capable of making trade-offs between cross-shard ratios and load balance, thereby consistently reducing both cross-shard traffic and load variance over time. Compared with the Random baseline, it achieves not only stable numerical improvements but also rare “plan–action–result” consistency in interpretability, which explains why strong LLMs can lead under extreme M-Stress3 conditions.

However, despite DeepSeek-R1’s strong decision-making logic and overall performance advantages in the Stress3 environment, its reasoning and execution still exhibit certain shortcomings. First, the model can sometimes be overly sensitive in shard assignments, especially when a shard simultaneously suffers from both high load and high cross-shard ratio (e.g., shard 6): on the one hand reducing traffic to mitigate congestion, while on the other hand raising weights to cut cross-shard costs, leading to unstable short-term allocations. Second, because migration actions are constrained by prefix granularity, DeepSeek-R1 occasionally introduces additional imbalance during the rebalancing phase, resulting in small oscillations in the reward curve after migrations. Finally, while the model’s rationales are logically clear, its long-term forecasting ability remains limited, showing insufficient anticipation of future hotspot shifts and relying instead on subsequent corrective adjustments to converge to better states. These issues suggest that even strong LLMs still have room for improvement when tackling the dynamic coupling of cross-shard efficiency and load balance, particularly in enhancing global consistency and long-horizon predictive capacity.

6 Related Work

Sharding has long been recognized as an important scalability path for blockchains. Early protocols such as Elastico[23], OmniLedger[14], and RapidChain[34] pioneered the idea of partitioning nodes and transactions into multiple shards to parallelize consensus and execution. More recent efforts, including BrokerChain[11] and SharPer[1], focus on improving cross-shard communication efficiency and secure routing. Despite these advances, two persistent bottlenecks remain: frequent cross-shard interactions inflate communication overhead, and load imbalance across shards undermines throughput and fairness[18, 33]. These challenges motivate the study of control mechanisms such as AP and AM.

AP in sharding blockchains aims to assign newly appearing user accounts to target shards in order to reduce cross-shard interactions while maintaining overall load balance. Early approaches were rule- or hash-based: for example, assigning accounts by address prefix, modulo hashing, or consistent hashing to evenly distribute them across shards [14]. Later, more sophisticated structure-aware placement strategies emerged, leveraging transaction or interaction statistics to colocate accounts that are highly connected with their introducers, thereby reducing immediate cross-shard costs [11]. Other works exploit community or graph structures without performing full graph repartitioning, instead using lightweight aggregated features to guide placement [13]. Spring [19] formulates AP as a deep reinforcement learning problem that jointly optimizes cross-shard ratio and load balance. Yet, its effectiveness depends on alignment between training distributions and live traffic, and

its stability under long horizons and strict feasibility constraints remains a practical challenge.

AM has been widely studied as a mechanism to sustain scalability in sharding blockchains by redistributing account states across shards [22]. Early efforts approached this problem through transaction-graph analysis, where transactions are modeled as edges between account nodes, and shard assignments are derived via graph partitioning or community detection techniques [3, 16]. To mitigate severe and prohibitive storage and computation overheads, community-aware partitioning has been proposed [17], aiming to colocate accounts with dense interactions to balance workload and reduce cross-shard transactions (CSTXs). However, these methods often struggle to simultaneously optimize load balance and CSTX minimization, as improvements in one metric may degrade the other. BrokerChain [11] introduces an intermediary broker network that coordinates account movements to reduce CSTXs, but this design raises centralization risks since brokers may become performance bottlenecks or single points of failure. Monoxide [31] adopts asynchronous consensus zones to scale execution capacity, indirectly alleviating migration pressure, while lock-based protocols [10] enforce safety during transfers by preventing race conditions and double spending. Although secure, lock-heavy protocols impose additional synchronization overhead, reducing throughput. LB-Chain [18] takes a predictive approach by using LSTM networks [9] to forecast workload distribution and proactively reallocate accounts. While effective at balancing shard loads, LB-Chain does not adequately address CSTX reduction, leaving communication overhead high. AERO [28] proposes a reinforcement-learning-driven migration framework that integrates specialized data structures for efficient state management.

The rapid progress of LLMs has opened new possibilities for complex decision-making tasks [24, 36]. Unlike traditional policies, LLMs can compress high-dimensional system states into textual prompts, reason over multiple objectives, and generate structured outputs with auditable rationales. Recent studies explore LLMs in domains such as cloud scheduling [6], distributed system optimization [30], and network management [7]. These works demonstrate that reasoning quality and model scale are key determinants of performance. To our knowledge, our work is the first to systematically benchmark LLM-driven schedulers for sharding blockchains, unifying AP and AM into a controlled decision environment and providing reproducible evaluation pipelines.

7 Conclusion

In this work, we presented ShardBenchmark, the first unified and auditable benchmark for evaluating AP, AM, and their joint scheduling in sharding blockchains. By designing controllable synthetic datasets that capture skew, stickiness, and community clustering, ShardBenchmark enables reproducible stress-testing of scheduling policies under diverse conditions. We further proposed an LLM-driven decision framework that transforms shard-level observations into structured actions with verifiable rationales. ShardBenchmark provides a standardized platform to rigorously study scheduling strategies for sharding blockchains. We believe it will foster future work on more advanced learning-driven policies and real-world deployment in high-throughput blockchain systems.

References

- [1] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. Sharper: Sharding permissioned blockchains over network clusters. In *Proceedings of the 2021 international conference on management of data*. 76–88.
- [2] Anthropic. 2024. The Claude 3 Model Family: Opus, Sonnet, Haiku. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbaf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf Accessed: 2025-09-26.
- [3] Charles-Edmond Bichot and Patrick Siarry. 2013. *Graph partitioning*. John Wiley & Sons.
- [4] Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blstein, Ori Ram, Dan Zhang, Evan Rosen, et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261* (2025).
- [5] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 international conference on management of data*. 123–140.
- [6] Sam Earle, Samyak Parajuli, and Andrzej Banburski-Fahy. 2025. DreamGarden: A Designer Assistant for Growing Games from a Single Prompt. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–19.
- [7] Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. 2025. From llm reasoning to autonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678* (2025).
- [8] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).
- [9] S Hochreiter. 1997. Long Short-term Memory. *Neural Computation MIT-Press* (1997).
- [10] Huawei Huang, Yue Lin, and Zibin Zheng. 2024. Account Migration across Blockchain Shards using Fine-tuned Lock Mechanism. In *IEEE International Conference on Computer Communications (INFOCOM)*. Vancouver, Canada, 20–23. *Corresponding author.
- [11] Huawei Huang, Xiaowen Peng, Jianzhou Zhan, Shenyang Zhang, Yue Lin, Zibin Zheng, and Song Guo. 2022. Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 1968–1977.
- [12] Huawei Huang, Guang Ye, Qinde Chen, Zhaokang Yin, Xiaofei Luo, Jianru Lin, Taotao Li, Qinglin Yang, and Zibin Zheng. 2023. BlockEmulator: An Emulator Enabling to Test Blockchain Sharding Protocols. *arXiv preprint arXiv:2311.03612* (2023).
- [13] Arijit Khan. 2022. Graph analysis of the ethereum blockchain data: A survey of datasets, methods, and future work. In *2022 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 250–257.
- [14] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. Omnidledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 583–598.
- [15] Michał Król, Onur Ascigil, Sergi Rene, Alberto Sonnino, Mustafa Al-Bassam, and Etienne Rivière. 2021. Shard scheduler: Object placement and migration in sharded account-based blockchains. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. 43–56.
- [16] Andrea Lancichinetti and Santo Fortunato. 2009. Community detection algorithms: a comparative analysis. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 80, 5 (2009), 056117.
- [17] Canlin Li, Huawei Huang, Yetong Zhao, Xiaowen Peng, Ruijie Yang, Zibin Zheng, and Song Guo. 2022. Achieving scalability and load balance across blockchain shards for state sharding. In *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 284–294.
- [18] Mingzhe Li, Wei Wang, and Jie Zhang. 2023. LB-Chain: Load-balanced and low-latency blockchain sharding via account migration. *IEEE Transactions on Parallel and Distributed Systems* 34, 10 (2023), 2797–2810.
- [19] 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 on Web Conference 2024*. 2836–2846.
- [20] Lu Liu, Sicong Zhou, Huawei Huang, and Zibin Zheng. 2021. From technology to society: An overview of blockchain-based DAO. *IEEE Open Journal of the Computer Society* 2 (2021), 204–215.
- [21] Xinmeng Liu, Haomeng Xie, Zheng Yan, and Xueqin Liang. 2023. A survey on blockchain sharding. *ISA transactions* 141 (2023), 30–43.
- [22] Yizhong Liu, Jianwei Liu, Marcos Antonio Vaz Salles, Zongyang Zhang, Tong Li, Bin Hu, Fritz Henglein, and Rongxing Lu. 2022. Building blocks of sharding blockchain systems: Concepts, approaches, and open problems. *Computer Science Review* 46 (2022), 100513.
- [23] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 17–30.
- [24] Sherwin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196* (2024).
- [25] OpenAI. 2025. Introducing GPT-4.1 in the API. <https://openai.com/index/gpt-4-1/> Accessed: 2025-09-26.
- [26] Brandon Liew Yi Quan, Nur Haliza Abdul Wahab, Arafat Al-Dhaqm, Ahmad Alshammari, Ali Aqarni, Shukor Abd Razak, and Koh Tieng Wei. 2024. Recent advances in sharding techniques for scalable blockchain networks: A review. *IEEE Access* (2024).
- [27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
- [28] Mingxuan Song, Pengze Li, Bohan Zhou, Shenglin Yin, Zhen Xiao, and Jieyi Long. 2025. AERO: Enhancing sharding blockchain via deep reinforcement learning for account migration. In *Proceedings of the ACM on Web Conference 2025*. 706–716.
- [29] Marios Touloupou, Marinos Themistocleous, Elias Iosif, and Klitos Christodoulou. 2022. A systematic literature review toward a blockchain benchmarking framework. *IEEE Access* 10 (2022), 70630–70644.
- [30] Jun Wang. 2025. A tutorial on llm reasoning: Relevant methods behind chatgpt 01. *arXiv preprint arXiv:2502.10867* (2025).
- [31] Jiaping Wang and Hao Wang. 2019. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th USENIX symposium on networked systems design and implementation (NSDI 19)*. 95–112.
- [32] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengan Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).
- [33] Shijiang Yuan, Jie Li, Jinghao Liang, Yuxuan Zhu, Xiang Yu, Jianping Chen, and Chentao Wu. 2021. Sharding for blockchain based mobile edge computing system: A deep reinforcement learning approach. In *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–6.
- [34] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 931–948.
- [35] Yuanzhe Zhang, Shirui Pan, and Jiangshan Yu. 2023. Txallo: Dynamic transaction allocation in sharded blockchain systems. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 721–733.
- [36] Xuanhe Zhou, Junxuan He, Wei Zhou, Haodong Chen, Zirui Tang, Haoyu Zhao, Xin Tong, Guoliang Li, Youmin Chen, Jun Zhou, et al. 2025. A Survey of LLM X DATA. *arXiv preprint arXiv:2505.18458* (2025).

A Notation Table

Symbol	Description	Symbol	Description
K	Number of Shards	G	Number of Communities
N_{old}	Pre-existing addresses	t	Block index
f	Prefix granularity	P	Migration period
t	New addresses	N	Total new addresses
a	Transactions per shard	c	Cross-shard transactions
v_j	Interaction count	V	Source matrix
M	Address mapping	s	Account count
γ	Discount factor	γ_r	Cross-shard penalty
k	Balancing term	τ	Balancing term, $\tau \in (0, 1)$
μ	Mean	σ	Standard deviation
ϵ	Stabilizer	ρ^1	State transition (AP)
ρ^2	State transition (AM)	S^1	State space (AP)
S^2	State space (AM)	\mathcal{A}^1	Action space (AP)
\mathcal{A}^2	Action space (AM)	R_t	Reward, $R_t = r_t^\alpha \cdot b_t^\beta$
r_t	Cross-shard efficiency	b_t	Load balancing
M_{AP}	MDP (AP)	M_{AM}	MDP (AM)
M_{AP+AM}	MDP (AP+AM)	p_t	Address proportion vector
$C(s_t)$	Feasible domain	O_t	Prefix-shard snapshot
$w_{t,p}$	Prefix-shard vector	$w(a)$	Base activity weight
$w(a)$	Normalized sender probability	y_a	Hotspot amplification
$\delta(\cdot)$	Indicator function	$Pr(R = r S = s)$	Conditional receiver probability
$p_{old}(r S = s)$	Existing-address receiver probability	$Unif(\cdot)$	Uniform distribution over a set
p_s	Stickiness	p_{new}	New address probability
$p_{old}(r S = s)$	Existing address probability	p_{intra}	Intra-community probability
$C(s_t)$	Community addresses	C	All addresses
p_t	Head-effect concentration	p_r	Reuse probability
T	Serialization operator	$\Phi_{AP}(s_t)$	AP compression
$\Phi_{AM}(s_t)$	AM compression	f_{θ}^{AP}	AP mapping
f_{θ}^{AM}	AM mapping	a_t^{AP}	AP action
a_t^{AM}	AM action	M_t	Migration pairs
$ M_t $	Number of migrations		

Table 5: List of Symbols

B Hyperparameters Settings

Table 6: Reward and MDP hyperparameters

Hyperparameter	Value
Cross-shard weight exponent, α	1
Balance weight exponent, β	1
Cross-shard nonlinearity, γ_r	3
Balance slope, k	3
Balance threshold, τ	0.5
Stabilizer, ε	1e-6
Discount factor, γ	0.99

Table 7: Task and control hyperparameters

Hyperparameter	Value
Number of shards, K	16
Communities, G	2000
Old accounts, N_{old}	50,000
Transactions (blocks \times txs), T	10^6 (1000 \times 1000)
Migration period, P	100
Max migrations per epoch, a	5
Prefix granularity, f	16 (hex prefixes)

Table 8: Synthetic data generation (global) hyperparameters

Hyperparameter	Value
Hotspot amplification, γ_a	4
Lognormal mean, μ	0.0
Lognormal std, σ	1.5
New-receiver prob., p_{new}	0.1
Intra-community prob., p_{intra}	0.75

C Overall Task Procedure

Algorithm 2: Address Placement

Input: Number of shards K ; hyperparameters $\alpha, \beta, k, \tau, \gamma_r$; transaction block stream

Output: Full-process logs \mathcal{L} and aggregated metrics

1 Initialize: Address mapping $\phi \leftarrow \emptyset$, shard account sets $\{\mathcal{S}_i\}_{i=0}^{K-1} \leftarrow \emptyset$

2 Load the first block and construct observation
 $s_0 = \{i, a, c, \ell, V, M, s\}$
// Iterative allocation and update

3 while data not exhausted do

4 Use policy π to produce action $a_t : \ell \rightarrow \{0, \dots, K-1\}$

5 For each $a \in \ell$, set $\phi(a) \leftarrow a_t(a)$ and update the corresponding $\mathcal{S}_{\phi(a)}$

6 obtain a, c under the updated ϕ

7 Compute $r_t, b_t, R_t = r_t^\alpha b_t^\beta$

8 Log $\{i, |\ell|, r_t, b_t, R_t\}$

9 load the next block and construct s_{t+1}

10 return \mathcal{L} and aggregated metrics

Algorithm 3: Account Migration

Input: Number of shards K ; migration period P ; hyperparameters $\alpha, \beta, k, \tau, \gamma_r$; transaction block stream

Output: Full-process logs \mathcal{L} and aggregated metrics

1 Initialize: Address mapping $\phi \leftarrow \emptyset$, shard account sets $\{\mathcal{S}_i\}_{i=0}^{K-1} \leftarrow \emptyset$

2 Load the first block and construct observation
 $s_0 = \{i, a, c, s, O\}$
// Iterative migration and update

3 while data not exhausted do

4 if $(i+1) \bmod P = 0$ **then**

5 Use policy π to produce up to a migrations
 $(u \rightarrow v, p)$

6 Apply migrations to ϕ and update $\{\mathcal{S}_i\}$

7 Recompute a, c under the updated ϕ

8 Compute $r_t, b_t, R_t = r_t^\alpha b_t^\beta$

9 Log $\{i, \text{migrated}, r_t, b_t, R_t\}$

10 Refresh O using the current block

11 Load the next block and construct s_{t+1}

12 return \mathcal{L} and aggregated metrics

Algorithm 4: AP+AM Unified Scheduling

Input: Number of shards K ; migration period P ; hyperparameters $\alpha, \beta, k, \tau, \gamma_r$; transaction block stream

Output: Full-process logs \mathcal{L} and aggregated metrics

1 Initialize: Address mapping $\phi \leftarrow \emptyset$, shard account sets $\{\mathcal{S}_i\}_{i=0}^{K-1} \leftarrow \emptyset$

2 Load the first block and construct observation s_0
// Iterative scheduling

3 while data not exhausted do

4 AP: Use policy π to produce placement a_t^{AP} (cover all new addresses) and apply

5 if $(i+1) \bmod P = 0$ **then**

6 AM: Use policy π to produce up to a migrations a_t^{AM} and apply

7 Recompute a, c under the updated mapping

8 Compute $r_t, b_t, R_t = r_t^\alpha b_t^\beta$

9 Log $\{i, |\ell|, r_t, b_t, R_t\}$

10 Build addr_old from the current block

11 Load the next block and construct s_{t+1}

12 return \mathcal{L} and aggregated metrics

Table 9: API version specifications

Model	API Version
Deepseek-R1	DeepSeek-R1
GPT-4.1	gpt-4.1-2025-04-14
Claude 3.7 Sonnet	gcp-claude37-sonnet
Gemini 2.5 Flash	gemini-2.5-flash-force-no-thinking

D Prompt-Response Illustration

To complement the case study in the main text, we provide here a full prompt and response used in the *M-Middle* dataset experiment with DeepSeek-R1. This example illustrates how the model reasons over shard statistics and outputs structured diversion proportions.

Instruction:

```
Task: Decide per-shard diversion proportions p = [p_0, ..., p_15] for routing new addresses so as to (1) reduce cross-shard traffic and (2) improve load balance.

You are given three length-16 integer arrays for the current block view:
- all_tx = [127, 126, 144, 122, 122, 120, 134, 124, 121, 97, 127, 156, 104, 116, 90, 170] # total tx count per shard
- cross_tx = [99, 106, 108, 86, 96, 84, 106, 84, 93, 63, 97, 122, 82, 84, 70, 116] # cross-shard tx count per shard
- new_cnt = [14, 19, 10, 11, 10, 13, 11, 7, 15, 5, 13, 13, 9, 8, 2, 19] # number of new addresses whose primary senders point to each shard i

**Semantics of p_i**
For new addresses whose primary sender shard is i, assign a fraction p_i to shard i to reduce cross-shard traffic. Higher p_i values significantly reduce the cross-shard transactions. The remaining fraction (1 - p_i) is diverted away from i to improve load balancing. You only output p; the executor performs the routing.

### Hard constraints (must satisfy)
1) Output a float list of length 16: [p_0, p_1, ..., p_15].
2) Each p_i in [0, 1].
```

Output format (must follow exactly)

- Put any reasoning ONLY inside <reasoning>...</reasoning>.
- Put the final list ONLY inside <answer>...</answer>.
- The list must be a valid Python list of floats of length 16, e.g.:

```
<answer>[0.55, 0.30, 0.80, ..., 0.65]</answer>
```

- No extra text inside <answer>.

```
<reasoning>
  Keep it in brief.
</reasoning>

<answer>
# Your final list here, length 16, e.g. [p_0, p_1, ..., p_15]
</answer>
```

Response:

```
<reasoning>
To determine the optimal diversion proportions, I need to balance two objectives:
```

1. Reduce cross-shard traffic: Higher p_i values keep more transactions within the same shard
2. Improve load balancing: Lower p_i values divert traffic away from overloaded shards

First, let me analyze the current state:

- Calculate the cross-shard ratio for each shard ($\text{cross_tx}[i]/\text{all_tx}[i]$)
- Identify which shards are overloaded (higher all_tx values)
- Identify which shards have high cross-shard traffic (higher cross_tx values)

For overloaded shards, I should set lower p_i to divert traffic away.

For shards with high cross-shard ratios, I should set higher p_i to keep more transactions within the shard.

Let me calculate the cross-shard ratio and normalize the load:

- Shard 0: $99/127 = 0.78$ cross-shard ratio, medium load
- Shard 1: $106/126 = 0.84$ cross-shard ratio, medium load
- Shard 2: $108/144 = 0.75$ cross-shard ratio, high load
- Shard 3: $86/122 = 0.70$ cross-shard ratio, medium load
- Shard 4: $96/122 = 0.79$ cross-shard ratio, medium load
- Shard 5: $84/120 = 0.70$ cross-shard ratio, medium load
- Shard 6: $106/134 = 0.79$ cross-shard ratio, high load
- Shard 7: $84/124 = 0.68$ cross-shard ratio, medium load
- Shard 8: $93/121 = 0.77$ cross-shard ratio, medium load
- Shard 9: $63/97 = 0.65$ cross-shard ratio, low load
- Shard 10: $97/127 = 0.76$ cross-shard ratio, medium load
- Shard 11: $122/156 = 0.78$ cross-shard ratio, very high load
- Shard 12: $82/104 = 0.79$ cross-shard ratio, low-medium load
- Shard 13: $84/116 = 0.72$ cross-shard ratio, medium load
- Shard 14: $70/90 = 0.78$ cross-shard ratio, low load
- Shard 15: $116/170 = 0.68$ cross-shard ratio, very high load

Based on this analysis:

- Shards with high load (2, 6, 11, 15) should have lower p_i values to divert traffic
- Shards with high cross-shard ratios (1, 4, 6, 12) should have higher p_i values to reduce cross-shard traffic
- Shards with low load (9, 14) can have higher p_i values to accept more traffic

Balancing these factors, I'll set p_i values:

- For high-load shards: lower values (0.3-0.5)
- For high cross-shard ratio shards: higher values (0.6-0.8)
- For low-load shards: higher values (0.7-0.9)
- For balanced shards: moderate values (0.5-0.7)

</reasoning>

```
<answer>[0.65, 0.6, 0.4, 0.65, 0.7, 0.65, 0.5, 0.65, 0.6, 0.8, 0.65, 0.35, 0.7, 0.65, 0.8, 0.3]</answer>
```