Anatomy of NEAR Sharded Blockchain: A Longitudinal Workload Characterization

Md. Ahsan Habib
Tulane University
New Orleans, LA, USA
mhabib@tulane.edu

Lu Peng
Tulane University
New Orleans, LA, USA
lpeng3@tulane.edu

Abstract

Blockchain sharding promises higher throughput, but production evidence on how real workloads interact with shard topology remains limited. We present a longitudinal characterization of the NEAR Protocol from 2020 to 2025 using historical on-chain data, analyzing both ecosystem structure and shard-level behavior across observed 4 to 9 shards configurations. We further construct five activity-specific interaction graphs and examine throughput, latency, receipt execution, gas usage, user and contract concentration, and cross-shard receipt flows. Our analysis shows that network activity is highly concentrated in a small set of applications and accounts, producing persistent shard imbalance and hub-like communication patterns. Although aggregate capacity increases with shard count, scaling is not monotonic under real workloads: throughput peaks at moderate shard counts, while cross-shard communication increases with finer partitioning and load remains uneven across shards. To explain these effects, we introduce two complementary shard-level models: a shard workload model that captures execution, origination, and communication structure, and a queueing-inspired effective demand model that relates throughput, gas usage, and cross-shard activity to latency. The effective demand metric strongly correlates with latency in moderate-shard regimes, but becomes less predictive at higher shard counts. This pattern is consistent with coordination overhead, rather than local queueing pressure, becoming the dominant influence on latency in that regime. These results show that sharded systems exhibit regime transitions under real workloads (throughput-limited, queueing-dominated, and coordination-dominated) rather than simple linear scaling with shard count. Our findings provide empirical guidance for workload-aware shard management and future sharding designs.

***Index Terms*—Sharded Blockchain, NEAR Protocol, Scalability, Graph-based Analysis, Queue-based Metric**

1. Introduction

Blockchain systems face fundamental scalability limitations due to the need for global consensus and largely sequential execution [23, 29]. Sharding has emerged as a promising Layer-1 (L1) solution that partitions state and computation across multiple shards, enabling parallel transaction processing [30]. While many sharding designs have been proposed, few production systems have deployed sharding at scale. Among them, the NEAR Protocol provides a unique opportunity to study the real-world behavior of a sharded blockchain over multiple years of operation [22].

Despite the growing adoption of sharding, an important question remains insufficiently understood: *does increasing the number of shards directly improve system performance, or is observed scalability primarily driven by workload evolution and application-level behavior?* Answering this question requires not only measuring throughput and latency, but also understanding how transactions, accounts, applications, and cross-shard communication interact with the sharding mechanism over time.

In this paper, we present a longitudinal workload characterization of the NEAR Protocol from 2020 to 2025. Using a dataset comprising billions of transactions, we construct five activity-specific graphs to capture value transfer, contract interactions, account management, and delegated execution. In parallel, we analyze shard-level dynamics across observed 4 to 9 shards configurations, examining throughput, workload distribution, and cross-shard communication patterns under realistic workloads.

Our analysis reveals that the scalability of NEAR is not solely determined by the number of shards, but is strongly influenced by application-layer concentration and communication structure. A small number of high-frequency accounts and applications dominate system activity, creating hub-and-spoke interaction patterns that lead to persistent shard imbalance. As a result, increasing the number of shards can improve capacity, but does not necessarily distribute workload evenly, and may increase cross-shard communication overhead under fine-grained partitioning.

To better understand these effects, we introduce a unified shard workload model that explains observed performance trends and identifies congestion hotspots, together with a queueing-inspired model that relates effective demand to latency. Our findings show that shard-level imbalance arises from application-driven workload skew and communication-heavy execution patterns rather than purely from system-level limitations, and that these effects lead to distinct operating regimes: sharded systems exhibit regime transitions under real workloads rather than simple linear scaling.

Contributions. This paper makes the following contributions:

- **Production-scale characterization of sharded blockchain workloads.** We analyze five years of NEAR transaction data, providing a large-scale empirical characterization of workload evolution, shard activity, and execution behavior in a deployed sharded blockchain. To the best of our knowledge, this is one of the first large-scale empirical studies of workload evolution in a production sharded blockchain.
- **Graph-based analysis of ecosystem structure.** We construct five interaction graphs capturing token flow, contract deployment, contract usage, account/key management, and delegated actions. These graphs reveal strong account- and contract-level concentration, where a few entities dominate network activity.
- **Empirical evidence of non-monotonic shard scalability.** We show that increasing shard count does not necessarily improve throughput or latency. Instead, workload skew, hot accounts, contract concentration, and cross-shard coordination limit parallel scalability.
- **Measurement-driven modeling of shard load and congestion.** We develop a unified workload model that captures execution load, transaction origination, locality, and cross-shard communication. We further derive an effective-demand metric linking shard-level activity to observed latency.
- **Implications for workload-aware sharding.** Our results motivate workload-aware shard management beyond uniform partitioning, including hotspot isolation, communication-aware placement, and adaptive reconfiguration.

2. Background

2.1. Blockchain Scalability Trilemma

Blockchain is often framed by the scalability trilemma [1], which posits that it can optimize only two of three properties: decentralization, security, and scalability. Decentralization refers to distributing control across many participants, security ensures resistance to attacks and ledger integrity, and scalability is the ability to handle increasing transaction volume efficiently. Early systems such as Bitcoin and Ethereum 1.0 prioritized decentralization and security, requiring all nodes to process every transaction, which limits performance [3, 17].

2.2. Sharding as a Scalability Solution

Sharding is a prominent L1 scaling approach that addresses the scalability trilemma by partitioning computation and storage across multiple node groups. Instead of requiring all nodes to process every transaction, the blockchain state and execution are divided into *shards*, each processing transactions in parallel, potentially increasing throughput with shard count and enabling horizontal scaling in which capacity grows with demand. Prominent systems such as Ethereum [2, 3] and Zilliqa [27] explore or implement sharding-based designs.

TABLE 1: NEAR Protocol actions and abbreviations.

Action Type	Abbr.	Action Type	Abbr.
transfer	`tr`	use_global_contract	`ugc`
function_call	`fc`	use_global_contract_by_account_id	`uga`
add_key	`ak`	deploy_contract	`dc`
delete_key	`dk`	deploy_global_contract	`dgc`
stake	`st`	deploy_global_contract_by_account_id	`dga`
create_account	`ca`	delegate_action	`dla`
delete_account	`da`		

2.3. NEAR Protocol: A Sharded L1 Blockchain

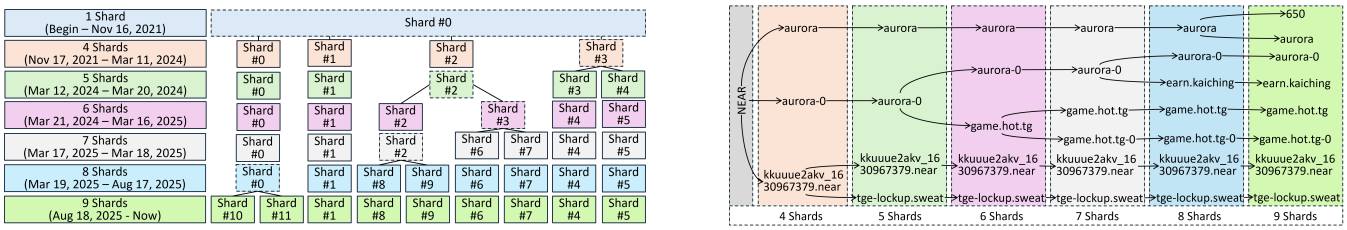
2.3.1. Account Model. NEAR uses a human-readable account model (e.g., `alice.near`) instead of cryptographic addresses, improving usability and enabling flexible key management. User and contract accounts share a unified address space, facilitating graph-based analysis. Account identifiers consist of 2–64 lowercase alphanumeric characters separated by dots [18]. NEAR supports multiple account types, including user, subaccount, contract, implicit (public key–derived), and protocol-level accounts. Each account can hold multiple access keys with distinct permissions, such as full-access and restricted function-call keys.

2.3.2. Nightshade Sharding. Nightshade is NEAR's novel approach to sharding, in which the system shards the state and computation but maintains a single main chain [25]. Each block on this chain contains information, called *chunks*, from all shards. A chunk represents the transactions and state transitions for a specific shard within a given block. In NEAR, chunks are produced by designated chunk producers-stake-weighted validators assigned per shard and rotated each epoch-while block producers are responsible for assembling blocks that reference the chunks from all shards.

2.3.3. Validators. NEAR uses a Proof-of-Stake (PoS) consensus mechanism, where validators stake tokens to produce and validate blocks and shard chunks [13, 24]. Validators are assigned to specific shards, validating only their chunks, which reduces cost and improves decentralization. Assignments rotate each epoch for load balancing and security. Malicious behavior results in slashing and reduced rewards. Validator accounts follow naming conventions: `.pool.near` and `.poolv1.near` denote public staking pools, while `.near` identifies the main validator account.

2.3.4. Actions. An action represents an executable operation within a transaction. In NEAR, a single transaction may contain multiple actions. We identify 13 action types, covering token transfer, contract execution, account/key management, contract deployment (including global variants), staking, and delegation. For brevity, we use abbreviated labels for each action type throughout the paper, the full list and descriptions are provided in Table 1.

2.3.5. Receipts. A Receipt is the internal message used by the NEAR Protocol to execute and track the progress of actions across the network. A single action may produce one or more receipts. Receipts are three kinds: `action, data`,



(a) Shard creation observed over time and their IDs.

(b) Static boundary accounts for shards.

Figure 1: Overview of shard creation and boundary accounts in the NEAR Protocol.

and `global_contract_distribution`. `Action` receipts execute one or more operations, such as token transfers or contract calls, while `data` receipts carry the results of these executed actions. `global_contract_distribution` receipts are system-level messages used to replicate or distribute global contracts across shards.

Example (Transaction, Action, and Receipt). Let Alice (`alice.near`) submit a transaction to swap tokens on a decentralized exchange (`dex.near`). Here, the transaction includes a `function_call` action invoking the swap. Actions generate receipts that are executed on the appropriate shards. In this example, the action creates `receipt #1` targeting `dex.near`, which executes the swap and produces `receipt #2` for a token transfer. `Receipt #2` executes on the token contract shard (e.g., `wbtc.near`), completing the transfer. Thus, a single transaction can result in multiple receipts, each consuming gas and recorded separately in `execution_outcomes`.

2.3.6. Contract Deployment. Developers write smart contracts on the NEAR blockchain using high-level languages such as Rust or AssemblyScript (a variant of TypeScript). These contracts are compiled into WebAssembly (WASM) bytecode. To deploy a contract on NEAR, the developer sends a transaction containing the compiled WASM code in the `dc` action to the target account. The target account becomes the contract account. It is important to note that a smart contract on NEAR can be deployed either by a user account or by another smart contract.

2.3.7. Shard Dynamics. In its original sharding roadmap published in September 2021 [20], NEAR outlined a four-phase transition toward a fully sharded blockchain.

Roadmap. Following the introduction of Simple Nightshade (Phase 0), which shards state but not processing and was planned for launch in November 2021, the roadmap proposed subsequent phases introducing shard-specific validation (Phase 1, expected 2022) and full state and execution sharding under Nightshade (Phase 2, expected 2023). The final phase, Dynamic Resharding (Phase 3), was envisioned for 2023 and aimed to enable automatic shard splitting and merging based on resource utilization, allowing the network to elastically adapt to workload fluctuations and achieve near-unbounded scalability [19].

Production Reality. The number of shards in the NEAR Protocol has evolved over time (Fig. 1a). The network operated as a single shard until November 2021, after which Simple Nightshade (Phase 0) introduced 4 shards with state partitioning but unified execution. Subsequent upgrades expanded the system to 5 and 6 shards in March 2024, followed by Nightshade 2.0 in August 2024, enabling stateless validation and full sharding. The network further scaled to 7, 8, and 9 shards in 2025 via incremental resharding, although fully dynamic resharding remains future work.

Shard assignment is determined by boundary accounts (Fig. 1b). When the shard count increases, an existing shard is split by introducing a new boundary, partitioning the account space lexicographically. For example, in the 4 shards configuration, boundaries such as `aurora`, `aurora-0`, and `kkuuue2akv_1630967379.near` define shard ranges. Subsequent expansions follow the same process, adding new boundaries (e.g., `tge-lockup.sweat`) to split existing shards and redistribute accounts.

3. Methodology

This study analyzes NEAR activity and performance in three stages: we construct activity-specific transaction graphs, perform longitudinal shard-level analysis across shard configurations, and characterize representative low, moderate, and heavy workload regimes.

3.1. Data Sources and Collection

We extract the full NEAR mainnet history from genesis to December 31, 2025 using the NEAR Public Data Lake on Google BigQuery [21]. The dataset contains 4.99B transactions (4,993,916,928) and 5.5B actions (5,503,152,301), spanning 272.8M accounts, including 272.7M users and 36.9K smart contracts. Although 92.7K contract deployment actions are observed, this exceeds the number of contracts due to NEAR supporting contract redeployment (i.e., code updates), unlike immutable deployments in Ethereum. The earliest block corresponds to height 9,820,210 (post regenesis, August 2020). Block heights are not strictly consecutive due to skipped, orphaned, or non-finalized blocks. Each epoch consists of 43,200 blocks, governing validator rotation and shard configuration.

3.2. Graph Construction Model

Table 3 summarizes the five graph types, their purposes, and the action kinds each covers: the token flow graph (TFG), the contract deployment graph (CDG), the contract usage graph (CUG), the account/key management graph (AMG), and the delegated protocol graph (DPG). Table 4 summarizes actions count of each category.

Four of the five are directed weighted graphs $G = (V, E, W)$, where V is the set of nodes (users or contracts), E is the set of directed edges (v_i, v_j), and W assigns a weight $w_{ij} \in \mathbb{N}_0$ to each edge. An edge $(v_i \to v_j)$ carries a graph-specific meaning and w_{ij} counts its occurrences: in the TFG, v_i interacts with v_j and w_{ij} is the interaction frequency; in the CUG, v_i invokes smart contract v_j and w_{ij} is the number of calls; in the AMG, v_i performs account or key management actions on v_j and w_{ij} is the number of such operations; and in the DPG, v_i performs delegated protocol actions on v_j and w_{ij} is the number of such actions. The CDG is unweighted, $CDG = (V, E)$ with $E = \{e_{ij} \mid (v_i, v_j)\}$, where each edge $(e_{ij} : v_i \to v_j)$ represents that node v_i deploys a smart contract v_j.

3.3. Shard Analysis Model

To assess the performance, scalability, and load distribution of the NEAR Protocol under its sharding mechanism Nightshade, we perform a longitudinal analysis of on-chain metrics. We define a sharding period P_k as the time during which the network operates with a constant number of k shards, and let $\mathcal{S}_k = \{s_1, s_2, \ldots, s_k\}$ be the set of active shards during P_k. For each NEAR epoch e within a period, we compute the metrics below for each shard $s_i \in \mathcal{S}_k$.

3.3.1. Throughput and Latency. We evaluate scalability by measuring system throughput and latency under increasing shard counts. Let $f_i(t)$ denote the number of transactions processed by shard s_i over interval t, so the total network workload is $f_{\text{net}}(t) = \sum_{s_i \in \mathcal{S}_k} f_i(t)$. We define shard-level throughput as $TPS_i(t) = \frac{f_i(t)}{t}$, and aggregate network throughput as $TPS_{\text{net}}(k) = \sum_{s_i \in \mathcal{S}_k} TPS_i(t)$. Transaction latency Lat is the on-chain inclusion-to-final-execution latency: the difference between the block timestamp of a transaction's final execution outcome and the block timestamp of the block that first included it. Measured at block-timestamp granularity, it captures shard queueing, execution, and cross-shard receipt propagation, and excludes pre-inclusion (mempool and submission) time, which is not observable from public on-chain data.

3.3.2. Shard Workload Characterization. To quantify shard-level workload, we adopt two complementary perspectives: computational workload and interaction-based workload.

Computational Workload (Gas-based). For each shard s_i over interval t, total execution cost is defined as $Gas_i(t) = \sum_{j=1}^{n_i(t)} g(r_{i,j})$, where $g(r_{i,j})$ denotes the gas cost by receipt $r_{i,j}$ and $n_i(t)$ is the number of executed receipts. We define average gas per receipt as $\overline{Gas}_i(t) = \frac{Gas_i(t)}{n_i(t)}$.

Interaction-Based Workload. We define an interaction matrix $M \in \mathbb{N}^{k \times k}$, where M_{ij} denotes the number of receipts originating from shard s_i and executed on shard s_j during interval t. From M, we derive the execution workload $E_i = \sum_{j=1}^{k} M_{ji}$, which captures the computational workload handled by shard s_i, and the originating workload $O_i = \sum_{j=1}^{k} M_{ij}$, which quantifies outbound receipt generation from shard s_i.

We further characterize communication behavior through the cross-shard receipt ratio $C_i = \frac{\sum_{j \neq i} M_{ji}}{E_i}$ and the intra-shard locality ratio $L_i = \frac{M_{ii}}{E_i} = 1 - C_i$, capturing the proportion of remote versus local executions. Let E_{tot} and O_{tot} denote the system-wide total execution and originating workloads, respectively. We then define the unified interaction-based workload metric: $\mathcal{W}_i = \alpha \frac{E_i}{E_{\text{tot}}} + \beta \frac{O_i}{O_{\text{tot}}} + \eta \left[\gamma C_i + (1-\gamma)(1-C_i)\right]$, where $\alpha, \beta, \eta \in [0, 1]$ satisfy $\alpha + \beta + \eta = 1$, and $\gamma \in [0, 1]$ controls the trade-off between cross-shard communication and intra-shard locality. In our experiments, we set $\alpha = 0.7$, $\beta = 0.1$, $\eta = 0.2$, and $\gamma = 0.6$, prioritizing execution workload while still accounting for receipt origination and communication patterns. These weights enter only $\mathcal{W}_i$. The effective-demand metric $\mathcal{D}_i^{\text{eff}}$ defined in Section 3.3.3 contains no free coefficients, so none of our latency, correlation, or regime results depend on them. A sweep of all four coefficients, released with our artifact, shows that within the execution-priority regime ($\alpha \in [0.5, 0.9]$, $\beta \in [0, 1-\alpha]$, $\gamma \in [0, 1]$, with E_i and O_i normalized by their totals) the shard-load ordering agrees with the default weighting at a median Kendall τ of 0.80–1.00 across all six configurations, though the identity of the single busiest shard is sensitive to γ.

3.3.3. Queueing-Based Interpretation of Shard Congestion. We model each shard s_i as a single-server queue, where transactions arrive at rate $\lambda_i(t)$ and are processed at service rate $\mu_i(t)$. The arrival rate is estimated as $\lambda_i(t) = f_i(t)/t$, where $f_i(t)$ is the number of executed transactions within interval t. We equivalently refer to this quantity as shard throughput, i.e., $\lambda_i(t) \approx \text{TPS}_i(t)$, and use it as a proxy for the effective arrival intensity.

In classical queueing theory, utilization is defined as $\rho_i(t) = \frac{\lambda_i(t)}{\mu_i(t)}$, and the average transaction latency $Lat_i(t)$ increases rapidly as utilization approaches system capacity. However, the service rate $\mu_i(t)$ is not directly observable from on-chain data. To address this, we approximate service demand using the average gas consumed per execution, $\overline{Gas}_i(t)$, leveraging the fact that gas cost on NEAR (a WASM-based runtime) reflects computational effort. Assuming comparable processing capacity across shards, we model the service rate as $\mu_i(t) \propto \frac{1}{\overline{Gas}_i(t)}$.

Based on this, we define a queue-inspired shard demand as $\mathcal{D}_i(t) = \text{TPS}_i(t) \times \overline{Gas}_i(t)$, which captures the combined

TABLE 2: NEAR shard scaling transitions and bottlenecks.

Transition	Driver	Cause	Bottleneck
4 → 6 shards	Consumer-app demand growth	Rapid `KAI-CHING` and `HOT` adoption; concentrated load on shard(s)	Execution/runtime pressure
6 → 8 shards	State and workload concentration	Resharding V3 splits (`game.hot.tg-0` and `earn.kaiching`)	Runtime/state + coordination
8 → 9 shards	Shard-0 state imbalance	Concentrated `.tg` and `users.kaiching` state; split fixed at `650`	State growth

TABLE 3: Graph construction methodology.

Type	Purpose	Action Kinds
TFG	Track flow of native tokens between accounts	`tr`, `st`
CDG	Record the (re)deployment of contracts	`dc`, `dgc`, `dga`
CUG	Capture interactions with contracts	`fc`, `ugc`, `uga`
AMG	Monitor account and key management actions	`ca`, `da`, `ak`, `dk`
DPG	Track actions executed on behalf of others	`dla`

TABLE 4: Number of transaction actions in each graph.

Type	TFG	CDG	CUG	AMG	DPG
Count	382,590,257	92,694	2,166,573,548	384,960,541	2,550,452,981

effect of transaction intensity and computational cost. While $\mathcal{D}_i(t)$ serves as a proxy for utilization under limited parallelism, it does not capture coordination overhead arising from cross-shard interactions. To incorporate this effect, we define an effective shard demand as $\mathcal{D}_i^{\text{eff}}(t) = \mathcal{D}_i(t) \times (1 + C_i)$, where C_i denotes the cross-shard interaction ratio of shard s_i. This formulation captures both execution-side demand and coordination overhead, effectively amplifying demand in the presence of cross-shard communication. Under the assumption of comparable shard processing capacity, $\mathcal{D}_i^{\text{eff}}(t)$ remains proportional to effective system utilization up to a scaling factor. Since it is unnormalized, larger values indicate higher overall system pressure and are expected to correspond to increased transaction latency $Lat_i(t)$. Note that $\mathcal{D}_i^{\text{eff}}$ contains no free weighting parameters: the coefficients $(\alpha, \beta, \eta, \gamma)$ appear only in the interaction-based workload metric $\mathcal{W}_i$ of Section 3.3.2.

Assumptions and validity. The queue-inspired model rests on three assumptions: (i) each shard behaves as a single-server queue; (ii) shards have comparable processing capacity, so that service rate is proportional across shards; and (iii) average gas per receipt is a valid proxy for service demand. The second follows from Nightshade's symmetric per-shard chunk production and identical per-shard gas limits, and the third from gas metering computational effort in the WASM runtime. Their limits are that validator CPU and memory, network delay, and scheduler state are not observable from public on-chain data. The model therefore captures *relative* execution pressure across shards rather than absolute service time, and we interpret its outputs accordingly.

3.3.4. User Distribution. We analyze total and active user distributions across shards to capture user concentration effects. Let $\mathcal{U}_i(t)$ denote the set of all users that have appeared in shard s_i up to time t. The active user set over interval t is defined as $\mathcal{U}_i^{\text{act}}(t) = \{u \mid \exists \tau \in tx_i(t) : u \in \{\text{sender}(\tau), \text{receiver}(\tau)\}\}$. The total and active user counts are given by $U_i(t) = |\mathcal{U}_i(t)|$ and $U_i^{\text{act}}(t) = |\mathcal{U}_i^{\text{act}}(t)|$, respectively. Inequality across shards $\mathcal{S}_k$ is quantified using the Gini coefficient [10], defined as $G(\mathbf{x}) = \frac{\sum_{i=1}^{k} \sum_{j=1}^{k} |x_i - x_j|}{2k^2 \bar{x}}$, where $\mathbf{x} = (x_1, \ldots, x_k)$ is a vector of shard-level quantities and $\bar{x}$ is their mean. In this context, $\mathbf{x}$ corresponds to either $(U_i(t))$ or $(U_i^{\text{act}}(t))$. A Gini coefficient of 0 indicates perfect equality across shards, while a value of 1 indicates maximum concentration in a single shard.

3.3.5. Contract Distribution. We further analyze contract activity across shards to capture workload concentration and diversity effects. Let $\mathcal{T}_i(t) = tx_i(t)$ denote the set of transactions of contract invocations in shard s_i over interval t, and let $\mathcal{C}_i(t) = \{\text{contract}(\tau) \mid \tau \in \mathcal{T}_i(t)\}$ denote the set of unique contracts invoked. The total number of contract invocations and the number of unique contracts are given by $T_i(t) = |\mathcal{T}_i(t)|$ and $N_i(t) = |\mathcal{C}_i(t)|$, respectively. Inequality across shards $\mathcal{S}_k$ is quantified using the Gini coefficient applied to $\mathbf{x} = (T_i(t))$ and $\mathbf{x} = (N_i(t))$, capturing workload concentration and contract diversity across shards. This distinction allows us to identify shards dominated by a small number of popular contracts versus those exhibiting a broader range of activity.

3.4. Trace-Driven Workload Characterization

3.4.1. Workload Construction. We construct a trace-driven workload characterization of the NEAR protocol using execution traces from real smart contracts and accounts collected between Jan 1, 2024 and Dec 31, 2025. To capture representative operating regimes, we select three one-week intervals based on system activity, including aggregate gas consumption and throughput measured at the transaction, action, and receipt levels. These correspond to *low* (Dec 26, 2023–Jan 1, 2024, 4 shards), *moderate* (Mar 12–18, 2025, 6/7 shards), and *heavy* (Mar 31–Apr 6, 2024, 6 shards) workloads. These intervals show distinct workload characteristics and are further supported by known variations in ecosystem activity (e.g., holiday slowdown and periods of elevated application demand). A detailed quantitative comparison of these metrics is provided in Section 4.4.

The *heavy* interval corresponds to peak HOT Protocol activity, comprising 65M+ transactions. The *moderate* interval represents a high-throughput and efficiency-oriented operating phase, processing 41M+ transactions. The *low* interval serves as a control scenario with 34M+ transactions, reflecting reduced user activity during a holiday period and dominated by essential infrastructure operations. Execution semantics, shard assignments, and transaction ordering are preserved within fixed weekly windows to ensure that the traces capture realistic shard-to-shard interactions and production-level system behavior.

3.4.2. Evaluation Metrics. System behavior under the characterized workloads is analyzed using throughput, end-to-end latency, gas consumption, cross-shard execution rate, and shard-to-shard execution interaction, which together capture both computational demand and inter-shard communication.

4. Result and Analysis

Although NEAR protocol account identifiers (users and contracts) range from 2 to 64 characters, we truncate addresses to their first 15 characters for space efficiency.

TABLE 5: Action flow distribution and dominant entities.

Category	Value	Key Insight
Total Action	5.50B	Large-scale ecosystem activity
U2U	59.11%	Dominant interaction type
U2C	34.55%	Strong contract engagement
C2C + C2U	6.35%	Limited contract-to-contract flow
Top User	1.13B (hotwallet.kaiching)	Extreme activity concentration
Top Contract	785.5M (game.hot.tg)	High-frequency application hub
Dominant	Relayer, Treasury, GameFi	Activity driven by few roles

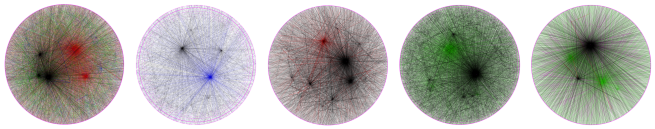
4.1. Global View of NEAR Ecosystem

We examine the NEAR ecosystem using graph-based representations of transaction actions. Figure 2 shows five interaction graphs, all exhibiting a common pattern: a small fraction of nodes form highly connected hubs. This is reinforced by the degree distributions in Fig. 3, which display heavy-tailed behavior, indicating strong participation heterogeneity. Most accounts have low activity, while a few dominate interactions. Notably, TFG shows node degrees exceeding 10^4, highlighting highly connected entities that mediate a large volume of actions. This concentration is driven by application-layer design rather than incidental user behavior. High-degree nodes typically correspond to custodial services, relayers, and large-scale applications that aggregate user activity through a small set of accounts. For example, the Kai-Ching ecosystem and Telegram-based applications (e.g., `game.hot.tg`) act as intermediaries, producing a hub-and-spoke structure in interaction graphs (Tables 5 and 7).

At the aggregate level, NEAR has processed 4.99B transactions, generating 5.5B actions and 21.8B receipts. As shown in Figure 4, the workload is dominated by delegated actions (`dla`) and contract calls (`fc`), indicating that activity is primarily application-driven rather than peer-to-peer. Despite the presence of multiple interaction types (Table 5), the workload is highly concentrated. The top 10 contracts account for over 84% of contract activity, while the top 10 users contribute about 68% of user actions (Table 6), showing that a small number of applications effectively control system activity.

This concentration has direct system implications: workload distribution is governed by dominant applications, shard imbalance is inherent rather than resolved by adding shards, and hub accounts amplify cross-shard communication, a mismatch between decentralized execution and highly centralized workload structure that motivates the shard-level analysis below.

Ecosystem Concentration. A small fraction of users and contracts dominates overall activity, creating inherent workload skew that propagates to shard-level imbalance.



(a) TFG (b) CDG (c) CUG (d) AMG (e) DPG

Figure 2: Graph visualizations of 20K sampled actions.

TABLE 6: Top smart contracts and infrastructure accounts by global activity frequency.

Type	Contract	Count	Type	User	Count
GameFi	game.hot.tg	785.8M	Treasury	hotwallet.kaich	1.13B
GameFi	earn.kaiching	292.7M	Relayer	relay.tg	734.0M
DeFi / FT	token.sweat	216.2M	Relayer	0-relay.hot.tg	672.8M
RetailTech	wallet.kaiching	200.1M	Registry	users.kaiching	530.6M
EVM	aurora	123.0M	Relayer	sweat-relayer.n	166.1M
Crowdwork	app.nearcrowd.n	112.4M	Relayer	relay.aurora	108.0M
Artifact	inscription.nea	73.3M	Relayer	sweat_welcome.	69.7M
DeFi / FT	wrap.near	31.4M	Relayer	spin.sweat	58.5M
GameFi	aa-harvest-moon	30.9M	Treasury	reserve.kaichin	45.7M
DeFi / DEX	v2.ref-finance.	25.7M	Oracle	oracle.sweat	45.2M

TABLE 7: Key impact entities and interaction characteristics.

Entity	Impact / Coverage	Role
kaiching ecosys	∼40% of total workload	Payment / rewards hub
game.hot.tg	14.2% of total workload	Transaction driver
reserve.kaichin	99.9% account coverage	Global interaction hub
sweat_welcome.	45.6% account coverage	Rewards service
oracle.sweat	9 connected accounts	Oracle service (special)
relay.aurora	16 connected accounts	Transaction relayer
spin.sweat	17 connected accounts	Limited-scope service
Top-1/3/10 User	20.5/46.1/64.6% of workload	User concentration
Top-1/3/10 Con.	14.3/23.5/35.8% of workload	Contract concentration

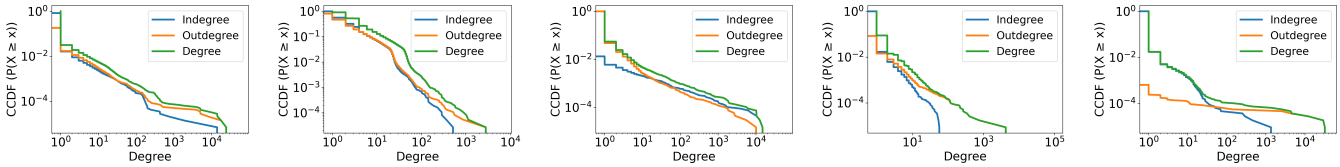
4.2. Analysis of Functional Graphs

To understand the mechanisms underlying the global concentration observed in Section 4.1, we analyze each functional graph in detail. Across all graph types, we observe consistent evidence that workload is not uniformly distributed, but instead driven by a small number of dominant entities with specialized roles.

4.2.1. Interaction Semantics. Table 8 reveals that interaction semantics fundamentally determine workload structure across the NEAR ecosystem. User-to-user (U2U) interactions dominate in TFG, AMG, and DPG graphs, consistently accounting for more than 95% of activity. This indicates that the majority of system workload is driven by direct user interactions rather than smart contract execution. In contrast, contract-centric graphs exhibit entirely different behavior. CDG graph is almost exclusively contract-to-contract (C2C), reflecting system-level or automated deployment processes, while CUG is dominated by user-to-contract (U2C) interactions, highlighting application-driven execution patterns.

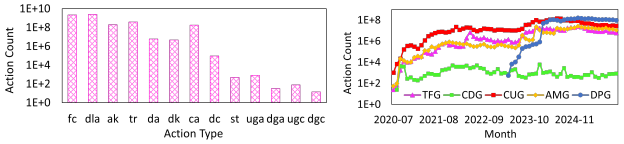
These results demonstrate that workload composition is highly heterogeneous across interaction types. User-dominated graphs (TFG, AMG, DPG) concentrate activity around a few high-degree user hubs, amplifying load skew. In contrast, contract-centric graphs (CUG, CDG) exhibit application-driven execution patterns, where a small set of contracts generates most operations. Overall, interaction types differ not only in semantics but also in how they concentrate and propagate system load.

4.2.2. Activity Concentration. Table 9 summarizes Top-k concentration across graphs, separating user- and contract-level activity. User activity exhibits extreme concentration in several graphs. In AMG, a single account contributes over 92% of interactions, while in DPG the top three users



(a) TFG (b) CDG (c) CUG (d) AMG (e) DPG

Figure 3: Node degree distributions (CCDF) of TFG, CDG, CUG, AMG, and DPG based on 150K sampled actions.



(a) Action count per type. (b) Action count per graph.

Figure 4: Log-scale visualization of total number of actions: (a) per action type; (b) per graph type over time.

TABLE 8: Interaction composition across NEAR graphs.

Graph	Total (share)	U2U (%)	C2U (%)	U2C (%)	C2C (%)
TFG	382.6M (7.3)	95.57	3.98	0.4	0.05
CDG	0.09M (∼0.0)	—	—	2.26	97.74
CUG	2.17B (41.4)	—	—	87.67	12.33
AMG	384.9M (7.4)	99.45	0.03	∼0.00	0.52
DPG	2.55B (48.9)	97.48	2.51	0.01	∼0.00

account for nearly 90%, reflecting strong reliance on a small set of relayer or treasury accounts. Contract concentration varies significantly across workloads. TFG and DPG show highly centralized contract activity, where the top contract contributes over 85% and 93%, respectively. In contrast, CDG remains fragmented on both user and contract sides, indicating the absence of dominant deployment entities. CUG presents a distinct application-driven pattern: user activity is relatively distributed, but contract activity is concentrated, with the top three contracts accounting for nearly 60% of interactions.

Interaction-Driven Skew. Workload imbalance is not uniform but depends on interaction semantics, where user and contract-driven activities produce distinct concentration patterns.

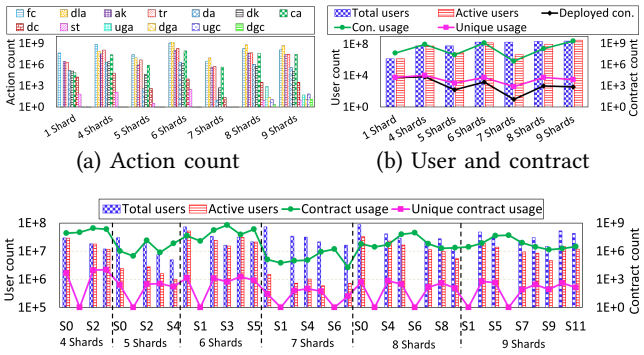
4.3. Shard-Level Dynamics

We analyze throughput, scalability, shard-wise load distribution, and account cardinality across all observed sharding periods P_k shown in Fig. 1a. The configuration set is $\mathcal{S} = \{\mathcal{S}_0, \mathcal{S}_4, \mathcal{S}_5, \mathcal{S}_6, \mathcal{S}_7, \mathcal{S}_8, \mathcal{S}_9\}$, where $\mathcal{S}_k$ denotes the system state with k shards, and $\mathcal{S}_0$ represents the pre-sharding (single-shard) stage. Among these, $\{\mathcal{S}_4, \mathcal{S}_6, \mathcal{S}_8, \mathcal{S}_9\}$ correspond to stable configurations, while $\mathcal{S}_5$ and $\mathcal{S}_7$ are short-lived transitional phases, included for the complete view of system evolution.

4.3.1. Action, User and Contract Distribution. Fig. 5 shows shard-level action usage, user counts, and contract deployment and invocation frequencies across configurations. The workload is heavily dominated by `fc` and `dla`,

TABLE 9: Top-k concentration across interaction graphs.

Graph	Type	Top Entity	Top-1 (%)	Top-3 (%)	Top-10 (%)
TFG	User	users.kaiching	48.13	75.63	76.83
	Con.	playember_rese	86.53	94.12	95.03
CDG	User	54dc30df	33.62	47.97	58.72
	Con.	0xfluxs	0.45	1.15	2.43
CUG	User	hotwallet.kaich	10.53	18.26	24.65
	Con.	game.hot.tg	36.27	59.75	87.23
AMG	User	users.kaiching	92.37	93.36	94.10
	Con.	pideky_escala	1.82	4.31	7.55
DPG	User	hotwallet.kaich	34.19	89.34	97.47
	Con.	playember_rese	93.11	99.82	99.85



(a) Action count (b) User and contract

(c) Per shard user and contract

Figure 5: Log-scale based visualization for different shard configurations: (a) action count distribution; (b) users (total and active) and contracts (deployed, usage and unique usage); (c) per shard users (total and active) and contracts (usage and unique usage).

which together account for 86% of all executions, while `ak`, `tr`, and `ca` form a secondary tier and other actions remain sparse (see Fig. 5a). As the number of shards increases, the system scales to hundreds of millions of users, with peak activity in the 6, 8, and 9 shards configurations. As expected, the 5 and 7 shards settings show unusually low active-user ratios. Contract behavior is similarly imbalanced, and Table 10 quantifies both effects: user allocation is uneven in every configuration (G = 0.38–0.50), contract usage is more concentrated still (G = 0.26–0.66), and the effective number of contract-bearing shards stays well below the configured count. Although the 4 shards configuration shows the broadest contract spread, the 6, 8, and 9 shards settings dominate total contract usage with markedly stronger concentration, indicating persistent shard-level hotspots rather than balanced scaling, and the user–contract

TABLE 10: Shard-distribution metrics for users and contract activity across configurations.

	Imbalance		Top-Shard Share		Eff. Shards		U-C Mismatch	
Config	User	Con.	User	Con.	User	Con.	Total	Active
4 shards	0.396	0.259	0.492	0.411	2.654	3.256	0.740	0.750
5 shards	0.482	0.588	0.489	0.618	2.814	2.123	1.070	0.881
6 shards	0.404	0.538	0.413	0.542	3.795	2.764	1.073	0.994
7 shards	0.489	0.642	0.413	0.572	3.797	2.409	1.515	1.501
8 shards	0.495	0.661	0.396	0.523	4.188	2.538	1.499	1.428
9 shards	0.379	0.614	0.214	0.414	6.216	3.200	1.330	1.260

U–C mismatch: misalignment between contract activity and user distribution across shards, measured separately using total users and active users.

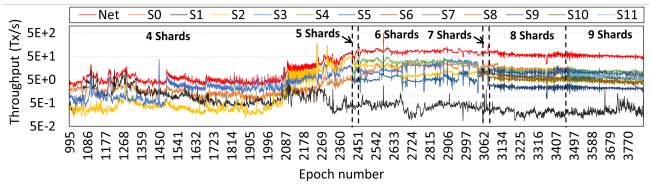
mismatch (0.74–1.52) shows that contract activity tracks neither total-user nor active-user placement.

4.3.2. Throughput and Latency. Across configurations, throughput and latency exhibit a non-monotonic scaling pattern as shown in Fig. 6. Mean network TPS rises from 11.75 (4 shards) to a peak of 84.21 (6 shards), then declines to 52.26 at 9 shards. Median TPS follows the same trend, peaking at 82.42 (6 shards) and stabilizing around 52–60 thereafter. Peak throughput is highly variable, reaching 484.1 at 6 shards before dropping to 85.5 at 9 shards. Latency shows the opposite trend, decreasing overall with scaling but with a sharp anomaly at 5 shards. Mean latency is 2.63 (4 shards), spikes to 11.69 (5 shards), and then steadily drops to 1.82 at 9 shards. Median latency decreases consistently from 2.15 (4 shards) to 1.80 (9 shards), while maximum latency remains highest at 5 shards (67.85), indicating transient congestion. Shard-sum consistency holds: mean shard-sum TPS closely matches network TPS (84.21 vs. 87.10 at 6 shards). Efficiency is best around 6 shards, while higher shard counts reduce both throughput gains and latency, suggesting diminishing returns from coordination overhead and workload imbalance. Normalizing by shard count separates per-shard capability from aggregate scaling: mean throughput per configured shard peaks at 14.0 Tx/s at 6 shards and falls to 5.8 Tx/s at 9, while the effective number of contract-bearing shards stays between 2.1 and 3.3 irrespective of the configured count (Table 10). The decline past 6 shards is therefore more consistent with the added shards being underused, alongside the cross-shard receipt ratio rising from 0.57 to 0.79, than with saturated per-node compute, which public on-chain data cannot observe directly.

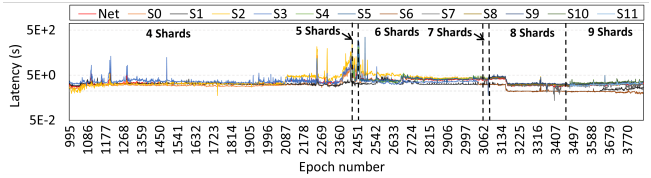
Non-Monotonic Scaling. Increasing shard count does not guarantee performance gains. Throughput peaks at moderate shard counts and declines as coordination overhead and workload imbalance increase.

4.3.3. Workload Analysis. Figure 7 shows shard-level execution volume, gas consumption, and interaction-based workload distribution.

Computational Workload. Execution is highly skewed across all configurations, with a small subset of shards processing the majority of transactions (e.g., $> 10^9$ executions in several shards under 6 to 8 shards settings), while others handle orders of magnitude fewer (Fig. 7a). This imbalance

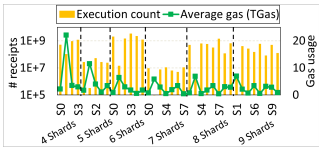


(a) Throughput for different shard configuration

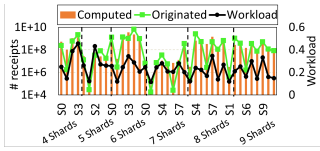


(b) Latency for different shard configuration

Figure 6: Log-scale based visualization over epoch number: (a) per shard transaction throughput and (b) per shard transaction latency.



(a) Receipt execution volume and average gas consumption



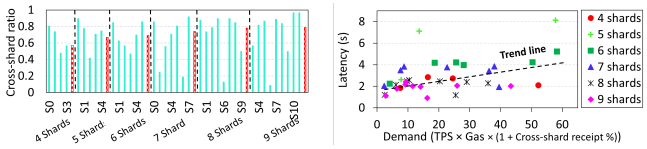
(b) Interaction-based workload distribution across shards

Figure 7: Shard-level performance characteristics: (a) receipt execution volume and mean gas consumption (Tera Gas) and (b) interaction-driven workload distribution.

persists regardless of shard count, indicating stable hotspot formation. Average gas consumption also varies widely, ranging from < 1 TGas to over 20 TGas. High-throughput shards tend to exhibit lower average gas, whereas low-throughput shards show higher gas intensity, suggesting a separation between lightweight bulk processing and heavier execution workloads.

Interaction-based Workload. Workload distribution remains highly skewed across all configurations (Fig. 7b), with a small subset of shards concentrating a large fraction of activity (e.g., up to ~ 0.45 in 9 shards and ~ 0.34 in 7 shards), while others remain lightly loaded (~ 0.01–~ 0.08). This imbalance evolves with system bottlenecks (Table 2). From 4 to 6 shards, scaling is driven by transaction volume, where hotspot activity (e.g., high-frequency wallets) triggers effective shard splitting. Between 6 and 8 shards, growth reflects broader ecosystem expansion, with workload concentration driven by high-frequency execution, signature verification, and state-processing activity. From 8 to 9 shards, the dominant constraint appears to shift toward state-size imbalance, particularly due to large account storage associated with `.tg` accounts, making state growth a limiting factor for further workload balancing.

4.3.4. Cross-Shard Imbalance. Table 11 summarizes shard-to-shard receipt-flow structure using the matrix M. The cross-shard receipt ratio $\overline{C}$ increases from 0.57 (4 shards) to 0.79 (9 shards), indicating that a larger fraction



(a) Shard and network level cross-shard receipt ratio

(b) Queue-inspired demand distribution across shards

Figure 8: Shard-level coordination and queue-derived demand: (a) cross-shard receipt ratio and (b) effective demand $\mathcal{D}_i^{\text{eff}}$ across shards.

TABLE 11: Flow-based shard interaction metrics.

Configs	# Receipts	$\overline{C}$	$CV(T)$	Asym	$\overline{HHI}_{\text{out}}$	Top Pair (Share)
4 shards	2.93B	0.57	0.83	0.58	0.37	S3 → S2 (0.40)
5 shards	140.68M	0.67	0.68	0.65	0.38	S3 → S2 (0.26)
6 shards	11.03B	0.69	0.72	0.68	0.34	S4 → S3 (0.27)
7 shards	57.10M	0.74	0.79	0.74	0.39	S3 → S6 (0.26)
8 shards	4.61B	0.78	0.85	0.71	0.37	S2 → S5 (0.37)
9 shards	2.48B	0.79	0.86	0.62	0.38	S1 → S4 (0.27)

$\overline{C}_{\text{net}}$: cross-shard receipt ratio; $CV(T)$: coefficient of variation across shards of total interaction load, where $T_i = E_i + O_i$; Asym: flow asymmetry across shard pairs; $\overline{HHI}_{\text{out}}$: average concentration of outgoing flows; Top Pair (Share): dominant shard pair and its fraction of total cross-shard flow.

of receipts becomes cross-shard as the system is partitioned more finely (see Fig 8a). Load dispersion $CV(T)$ remains high in every configuration and peaks at 8–9 shards (0.85–0.86), indicating stronger imbalance in aggregate shard activity at higher shard counts. Directional imbalance (Asym) remains substantial across all settings (0.58–0.74), with a local maximum at 7 shards and a lower value at 9 shards, implying persistent but non-monotonic one-way flow skew between shard pairs. Flow concentration is comparatively stable: mean HHI_{out} stays within 0.34–0.39, a moderate concentration of outgoing destinations per source shard. The top-pair contribution ranges from 0.26 to 0.40, showing that a single shard pair can account for a non-trivial share of cross-shard traffic, with the strongest dominance observed in the 4 shards configuration.

4.3.5. Queueing vs. Coordination Effects. The proposed effective demand metric $\mathcal{D}_i^{\text{eff}}(t)$ correlates strongly with transaction latency in moderate shard configurations (5–6 shards), where the system operates under congestion-like conditions consistent with single-server queueing behavior: $r = 0.899$ at 5 shards ($p = 0.038$, 95% CI $[0.08, 0.99]$) and $r = 0.834$ at 6 shards ($p = 0.039$, CI $[0.07, 0.98]$), indicating that latency is largely driven by local execution pressure. Because each aggregate correlation is computed across shards within a configuration, n equals the shard count (4–9). The intervals are correspondingly wide and we report them in full rather than summarizing them. Rank-based correlations agree in sign and ordering, with Spearman $\rho = 0.70$ at 5 shards ($p = 0.19$) and $\rho = 0.83$ at 6 shards ($p = 0.042$), against $\rho = 0.00$, 0.10, and 0.15 at 7, 8, and 9 shards. At $n = 5$ a rank test has almost no resolution, since the smallest attainable two-sided p is 0.017, so the 5-shard aggregate rests on the per-epoch evidence below rather than on its own significance.

However, as the number of shards increases, the correlation between $\mathcal{D}_i^{\text{eff}}(t)$ and latency weakens significantly, indicating a transition in the dominant performance bottlenecks. In highly sharded configurations, the correlation drops sharply and loses significance (7 shards: $r = 0.025$, $p = 0.96$; 8 shards: $r = 0.148$, $p = 0.73$; 9 shards: $r = 0.208$, $p = 0.59$), suggesting that latency is increasingly governed by distributed system effects, including cross-shard communication, asynchronous receipt processing, and scheduling overhead, rather than local queueing pressure alone.

To test whether this transition survives a larger sample, we recompute the same across-shard correlation *within each epoch*, yielding one correlation per epoch. The contrast is pronounced: the mean per-epoch correlation is 0.89 at 5 shards and 0.42 at 6 shards (622 epochs), but falls to 0.13 and 0.20 at 8 and 9 shards (415 and 373 epochs). Statistical significance behaves accordingly, with 71% of 5-shard epochs and 30% of 6-shard epochs individually significant, against none at 8 or 9 shards. This significance count should be read with care: each epoch's correlation uses only $n = 8$–9 shards, so detecting a correlation of 0.2 at the 5% level is not possible in principle, and the absence of significant epochs at high shard counts reflects limited statistical power as much as a weak relationship. The effect-size contrast is the more informative comparison, and it is if anything understated, since the 5-shard configuration clears a *higher* significance threshold ($|r| \geq 0.88$ at $n = 5$, versus 0.71 at $n = 8$) in the majority of its epochs. The 6-shard association is nonetheless weaker per epoch than its aggregate $r = 0.834$ suggests, so we treat the moderate-shard regime as anchored at 5 shards.

At low shard counts (e.g., 4 shards), latency behavior exhibits a different pattern, where transaction throughput shows strong correlation with latency ($r = 0.852$), while demand-based metrics are less predictive. We note that this 4-shard throughput–latency correlation is not statistically significant at $n = 4$ ($p = 0.15$) and is reported as descriptive. This indicates that system performance in this regime is primarily constrained by throughput capacity rather than workload composition. Figure 8b shows the scenario. These results reveal distinct operating regimes in sharded systems: throughput-limited at low shard counts, queueing-dominated at moderate levels, and coordination-dominated at high shard counts. While the effective demand metric captures both execution pressure and coordination overhead, it becomes less predictive when coordination effects dominate.

Separating shard count from workload evolution. Because shard count and workload both change along a single timeline, we isolate the effect of resharding by comparing matched windows of equal length immediately before and after each boundary, which holds the workload approximately fixed across the change. Most boundaries are unusable for this purpose: the outgoing configuration is already congested when the reshard occurs, so the pre-window measures transition congestion rather than steady state. We therefore admit a boundary only when its pre-window

latency lies at or below the outgoing configuration's steady-state mean. Two boundaries satisfy this, $6 \rightarrow 7$ and $8 \rightarrow 9$. The $8 \rightarrow 9$ transition is the cleanest, with a pre-window latency of 1.61 s against an 8-shard steady-state mean of 2.09 s. Across that boundary, latency barely moves ($1.61 \rightarrow 1.72$ s, a factor of 1.07) and network throughput is essentially unchanged ($58.7 \rightarrow 57.9$ Tx/s, a factor of 0.99). The result is stable for window widths from 5 to 30 epochs. Since each resharding ships with a protocol upgrade, this bounds the combined short-window effect of the shard-count change and the upgrade together. This null result does not contradict our non-monotonic throughput finding. It explains it, because beyond roughly six shards, additional shards no longer translate into observed throughput, and latency ceases to track local demand.

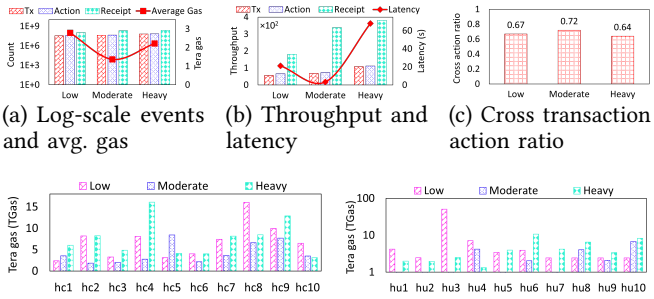
We also note that the extreme latencies visible in the per-configuration series belong to two short-lived transitional configurations rather than stable operating points. The 5- and 7-shard configurations persisted for only 14 and 4 epochs respectively (roughly 8 and 2 days), against steady-state means of 2.63, 4.00, 2.09, and 1.82 s for the 4-, 6-, 8-, and 9-shard configurations.

Interpretation and confounding factors. We frame the high-shard regime as *coordination-dominated* on the basis of two converging observations: the demand–latency relationship weakens and loses significance, while the cross-shard receipt ratio rises monotonically to 0.79 and is highest exactly in that regime (Figure 8a). Scheduling and coordination overhead are not directly observable from public on-chain data, so this remains an interpretation supported by the evidence rather than a conclusively established cause. Four factors qualify it. First, shard count is confounded with workload evolution, since NEAR's reshardings occur along a single timeline and its workload grew over the same period, so cross-era comparisons are associational. Second, each resharding shipped alongside a protocol upgrade, so any boundary comparison bounds the combined effect of the two rather than isolating shard count. Third, the aggregate correlations rest on $n = 4$–9 shards per configuration, which limits both interval precision and per-epoch statistical power. Fourth, validator CPU, memory, and network delay are not exposed by public data, so we cannot separate per-node compute limits from coordination overhead directly, and instead rely on shard-normalized proxies.

Regime Transition. Shard behavior shifts from throughput-limited to queueing-dominated and finally to coordination-limited regimes, fundamentally changing the drivers of system performance.

4.4. Performance Under Trace-driven Workloads

Figure 9a illustrates the absolute transaction volume, action count, receipt executions, and the average gas consumed per execution under *low*, *moderate* and *heavy* workloads. Compared to the *low* regime, the *heavy* regime experiences nearly a 2× increase in transaction volume, a 1.7× increase in actions, and more than a 2× increase in



(a) Log-scale events and avg. gas (b) Throughput and latency (c) Cross transaction action ratio

(d) Hot contracts' avg. gas usage (e) Hot user' avg. gas usage

Figure 9: Comparison of *low*, *moderate*, and *heavy* workload weeks across event counts, throughput/latency, cross-shard ratio, and gas usage for high-activity contracts and users (*hc*: hot contract, *hu*: hot user).

receipt executions. Despite the higher workload intensity, the average gas consumed per receipt decreases by about 20%, indicating a shift toward lower-cost receipts on average. The *moderate* regime lies between these two extremes, with approximately 21% higher transaction volume, 9% higher action count, and 87% increased receipt executions compared to the *low* regime. Interestingly, the average gas per receipt in this regime is lower than in the *low* regime. During this interval (Mar 12–18, 2025), the NEAR ecosystem hosted development initiatives, including a hackathon focused on useful agent applications (March 14, 2025) [9]. These events likely contributed to elevated developer engagement and increased volumes of relatively simple receipts, consistent with the observed lower average gas usage in the moderate workload.

As shown in Fig. 9b, throughput increases steadily from the *low* to the *heavy* workload across transactions, actions, and receipts. However, this improvement comes at the cost of a sharp rise in latency under *heavy* load, while the *moderate* regime maintains relatively high throughput with substantially lower latency. Figure 9c shows the cross-shard action ratios across workload regimes. The *moderate* workload exhibits the highest cross-shard ratio, reflecting its 7 shards configuration, while the *low* and *heavy* regimes (from 4 and 6 shards periods) show slightly lower but comparable levels. We further analyze hot contracts and users and their gas consumption (Figs. 9d and 9e). Hot contracts consistently consume more gas and show higher variance than hot users across all regimes. The *moderate* workload is the most balanced, whereas the *heavy* regime amplifies gas usage and variability, particularly for contracts, while hot users remain relatively stable with few high-cost outliers.

5. Related Work

A number of prior work has analyzed Ethereum transaction activity using graph-based methods, focusing on transaction relationships, ERC20 token transfers, Ether flows, and smart contract interactions. Chan et al. [5] applied graph analysis to Ethereum transactions without capturing complete transactional semantics; Chen et al. [7] added

anomaly detection over a detailed analysis of transaction data; and Guo et al. [11] examined transaction relationships from a network science perspective. Lin et al. [16] modeled transactions as a temporal weighted multi-digraph with dedicated sampling and representation techniques, while Chen et al. [8] and Somin et al. [26] characterized the ERC20 token ecosystem and its transfer contracts. Kiffer et al. [14] studied contract creation and invocation patterns without employing graph-based analysis, Charlier et al. [6] built contract-level Ether-transfer graphs at relatively small scale, and Cachin et al. [4] compared Bitcoin, Ethereum, and Hyperledger Fabric without focusing on workload or sharding characteristics.

A parallel line of work studies sharded-system design and workload-aware partitioning. OmniLedger [15] and Monoxide [28] address secure cross-shard transaction processing and throughput scale-out, while BrokerChain [12] and graph- and account-partitioning shard-placement methods such as TxAllo [31] reduce cross-shard traffic and hot-shard imbalance by reassigning accounts across shards. These works evaluate largely on prototypes or in simulation. To the best of our knowledge, no prior work has conducted a comprehensive transaction-level and graph-based analysis of a production sharded blockchain. Our contribution is complementary: we measure how the phenomena these designs target (cross-shard receipts, hot shards, and workload imbalance) actually manifest in NEAR across multiple years and six successive shard configurations.

6. Discussion

Load metrics as congestion indicators. The workload metric captures execution, origination, and communication effects, while the effective-demand metric relates throughput, gas usage, and cross-shard activity to latency. Both are most informative in moderate shard configurations, where higher demand correlates with higher latency. At higher shard counts the relationship weakens, so execution-side metrics alone become insufficient. They identify stressed shards and motivate workload-aware resharding and hotspot-isolation strategies keyed to active accounts, receipt origination, contract hotspots, and shard-to-shard communication rather than shard size, though they do not by themselves establish that any policy will improve performance. Static or random shard assignment is therefore insufficient, and three design implications follow.

Adding shards is not, by itself, a scaling mechanism. Aggregate throughput peaks near six shards and declines thereafter, while the cross-shard receipt ratio rises to 0.79: each additional shard converts previously local work into cross-shard receipts, and beyond a workload-dependent point the coordination it creates outweighs the parallelism it adds. Shard count is a parameter with an empirical optimum for a given workload, not a monotonic scaling knob.

Placement matters more than partition count. Execution and origination load are highly skewed, and a handful of contracts and accounts generate most of the traffic. Hotspot-aware placement (assigning frequently interacting accounts to the same shard, and splitting genuinely hot contracts) targets the dominant cost directly, whereas uniform hashing does not, and resharding should be triggered by observed load imbalance rather than a fixed schedule. Symmetric provisioning compounds the skew: identical gas limits and validator resources per shard leave the busy shards binding while the idle ones are over-resourced, so asymmetric provisioning would raise effective capacity without changing the partition, at the cost of a more complex validator-assignment mechanism.

Cross-shard receipt handling is the highest-value optimization target. Coordination cost grows with the cross-shard ratio, which rises with shard count, so hardware and runtime effort is best directed at the receipt path: faster signature verification, cheaper state access for receipt application, and lower latency in asynchronous receipt propagation. These benefits grow precisely in the regime where added shards stop helping.

Limitations. Our analysis relies on public on-chain data and cannot directly observe validator CPU utilization, memory pressure, network delay, or scheduler behavior, so hardware and runtime bottlenecks are inferred from workload, gas, state, and communication patterns rather than measured. The confounds bounding our causal claims are detailed in Section 4.3.5: shard count co-varies with workload evolution and with the protocol upgrade shipped alongside each resharding, and the across-shard correlations rest on $n = 4$–9 observations per configuration. We accordingly present the coordination-dominated regime as an interpretation consistent with the evidence rather than an established cause.

7. Conclusion

We presented a longitudinal workload characterization of the NEAR Protocol combining graph-based ecosystem analysis with shard-level performance measurements. Across the observed 4- to 9-shard configurations, activity is highly concentrated in a small number of applications and accounts, shard utilization remains uneven, and cross-shard communication increases with finer partitioning. A queue-derived effective-demand metric is strongly associated with latency in moderate-shard regimes, but becomes less informative at higher shard counts. This pattern is consistent with coordination overhead, rather than local queueing pressure, becoming the dominant influence on latency in that regime. Observed scalability in production sharded blockchains therefore depends not only on shard count, but also on workload concentration and communication structure, providing an empirical basis for future workload-aware studies of shard management, communication overhead, and performance optimization.

Acknowledgement

This work is supported in part by an NSF Grant 2618897. We appreciate the computational resources provided by the Louisiana Optical Network Infrastructure (LONI) and valuable suggestions from anonymous reviewers.

Artifact Appendix

Abstract

The artifact contains the BigQuery queries used to extract the NEAR mainnet measurements analyzed in this paper, the derived per-epoch and per-shard metrics, and the Python scripts that reproduce the paper's statistical results and figures. Specifically, it regenerates the aggregate and per-epoch demand–latency correlations of Section 4.4, the pre/post-resharding window comparison, the per-configuration throughput and latency figures, and the weight-sensitivity analysis of the flow-based load metric.

The artifact checks itself. A single script, `verify.py`, recomputes all 40 quantitative claims the paper makes, compares each against the value printed in the paper at the precision the paper quotes it, prints a PASS/FAIL table, and exits non-zero if any check fails. A reviewer can therefore establish reproduction with one command in under a minute, rather than by reading numbers off console output.

All derived data required to reproduce every reported number is included, so no BigQuery account, credentials, network access, or database is needed. Evaluation requires only a Python 3.12 environment, and the full evaluation completes in under two minutes on a commodity laptop.

Artifact check-list (meta-information)

- **Program:** Python 3.12 analysis and plotting scripts; 208 BigQuery SQL queries.
- **Data set:** Derived metrics from the NEAR mainnet public dataset on Google BigQuery (`bigquery-public-data.crypto_near_mainnet_us`), genesis through 31 December 2025; included in the artifact.
- **Run-time environment:** Any OS with Python 3.12; no network access, credentials, or database required. A `Dockerfile` is provided.
- **Hardware:** Any commodity machine; no GPU or special hardware.
- **Metrics:** Pearson correlation with two-sided p-values and Fisher-z 95% confidence intervals; Kendall τ rank agreement; mean network throughput and latency per shard configuration.
- **Output:** Console tables, CSV files under `results/`, and PNG figures under `Figure/output/` and `Figure/Load/output/`.
- **Experiments:** Seven experiments, run individually or by one driver script; no parameter tuning required.
- **How much disk space required?** 5 MB for the bundle; approximately 500 MB once the Python dependencies are installed.
- **How much time is needed to prepare workflow?** Under 5 minutes.
- **How much time is needed to complete experiments?** 82 s for everything; 3 s for the `--quick` verification path.
- **Publicly available?** Yes.
- **Code licenses:** MIT (code); CC BY 4.0 (derived data).
- **Archived DOI:** `10.5281/zenodo.21843777` (v2.0, the evaluated version; `10.5281/zenodo.21828027` resolves to the latest)
- **Badges claimed:** Available, Reviewed, Reproducible.

Description

How to access. The artifact is archived at https://doi.org/10.5281/zenodo.21843777. This version-specific DOI pins exactly the bundle described here, while the concept DOI `10.5281/zenodo.21828027` always resolves to the most recent version.

Hardware dependencies. None. All timings quoted below are wall-clock on an Intel Core i7-14700F with 32 GB of RAM running Python 3.12.3. The workload is single-threaded and modest, so a slower machine changes the numbers but not the outcome.

Software dependencies. Python 3.12 with `pandas`, `numpy`, `scipy`, `openpyxl`, and `matplotlib`. `requirements.txt` pins the versions used for the paper. As noted below, verification also passes under a substantially different set, so the results do not hinge on them.

Data sets. All derived data is included. `Figure/Fig_all_new.xlsx` holds the per-epoch per-shard throughput and latency series and the period-aggregate metrics. `Figure/Load/shard_load_inputs.csv` holds the per-shard execution, origination, and cross-shard inputs to the workload metric. The `.sql` files reproduce the original BigQuery extraction. Re-running them is optional and requires a billed Google Cloud project.

Installation

```
python -m venv .venv
. .venv/bin/activate
pip install -r requirements.txt
```

On Windows, activate with `.venv\Scripts\activate` instead. All scripts resolve their inputs relative to their own location and may be run from any working directory. Alternatively, `docker build -t near-iiswc26 .` builds a container that runs the full verification as a build step, so a successful build is itself evidence of reproduction. A `Makefile` wraps the same entry points, and nothing in the artifact requires `make`.

Experiment workflow

One command runs every experiment, writes each result to `results/`, regenerates all figures, and finishes by verifying the numbers against the paper:

```
python run_all.py            # 82 s
python run_all.py --quick    # 20 s
```

Steps are independent. A failure in one is reported at the end with its exit status and does not prevent the others from running. Every experiment can also be run on its own, in any order, as listed in Table 12.

Evaluation and expected results

The primary evaluation is automated:

```
python verify.py          # 40 checks, 35 s
python verify.py --quick  # 33 checks, 3 s
python verify.py --list   # checks, tolerances, refs
```

Each check carries an absolute tolerance matched to the precision at which the paper quotes the value, so a figure

TABLE 12: Experiments, the claims they support, and expected results. Check IDs are the labels `verify.py` prints. Scripts live in `Figure/`, except `sensitivity_sweep` and `plot_load_heatmaps`, which are in `Figure/Load/`.

ID	Script	Expected result
E1	`queue_correlations.py`	Demand–latency r (and p): 5 shards 0.8991 (0.0379); 6 shards 0.8336 (0.0392); 7–9 shards 0.0253, 0.1477, 0.2079 (0.957, 0.727, 0.592). Exactly two configurations significant at $p < 0.05$.
E2	`perepoch_correlations.py`	Mean per-epoch r: 0.891 (5 shards, 14 epochs, 71.4% individually significant); 0.133 (8 shards, 415 epochs); 0.198 (9 shards, 373 epochs); 0% significant at 8 and 9 shards.
E3	`resharding_windows.py --window 15`	Steady-state mean latency 2.63, 4.00, 2.09, 1.82 s for 4, 6, 8, 9 shards. 8 → 9: latency ratio 1.07, throughput ratio 0.99. Exactly two boundaries flagged `clean` (6 → 7, 8 → 9). `throughput_by_section.py` agrees by an independent route.
E4	`sensitivity_sweep.py`	Median Kendall τ vs. default weights: 1.000, 0.800, 0.867, 0.810, 0.857, 0.889 for 4–9 shards; range 0.80–1.00, floor at 5 shards.
E5	`plot_throughput_latency.py`; `plot_load_heatmaps.py`	Twelve per-configuration throughput and latency figures; six load-sensitivity heatmaps at $\gamma = 0.6$. Compared visually against the paper.
E6	`near_data.py`	The documented data correction relabels 415 8-shard-era S2 latency values as S4; none remain afterwards.
E7	`extract_partition_metrics.py`; `s2s_top5_metrics_table.py`	Partition-balance metrics per configuration; the shard-interaction table emitted as LaTeX. Descriptive tables, not tested claims.

printed to four decimals is checked to $\pm 5 \times 10^{-4}$, one quoted as "approximately 2.6 s" to $\pm 5 \times 10^{-3}$, and counts exactly. Tolerances are not padded to make checks pass. Expected output ends with `40/40 checks passed` and `RESULT: artifact reproduces every checked claim in the paper.` We have confirmed this from a clean extraction of the archived bundle into a fresh virtual environment.

Table 12 maps each experiment to the paper claim it supports and the values to expect. `CLAIMS.md` in the bundle gives the same map at full precision, per check.

Experiment customization and reuse

The artifact is built to be extended, and `EXTENDING.md` documents how.

Every analysis exposes a function returning a `DataFrame` separately from the `main()` that prints it (`compute_perepoch`, `compute_windows`, `compute_sensitivity`, `compute_correlations`), so the pipeline can be imported and reused rather than re-implemented. `verify.py` is itself written against this API and serves as the worked example. New analyses are wired into the automation by appending one entry to the step list in `run_all.py` and one four-line `Check` to `verify.py`. Both registries are plain lists, with nothing discovered by reflection.

The 208 SQL queries are usable without any of our Python: they encode how to obtain per-shard throughput, latency, gas, cross-shard receipt flows, and hot contract/user activity from NEAR's public dataset, including the parts that are easy to get wrong. Extending the study past 31 December 2025 means adjusting the date predicates in the per-era directories, not rewriting queries.

The flow-based workload metric is chain-agnostic. It requires only per-shard execution load E_i, originated load O_i, and cross-shard fraction C_i. Supplying a CSV with those columns makes both the sweep and the heatmap scripts work unchanged on another sharded system. The weight sweep accepts `--quick` for a coarse grid that reproduces all six quoted medians exactly at roughly 1/30 of the cost, and the grid resolution is settable, so the stability result can be re-examined at any resolution.

Notes

Shard identifiers are not contiguous across configurations, because NEAR reuses and renumbers shard IDs at each resharding. Scripts therefore segment the time series by *which* shards are active rather than by how many. Grouping by shard count instead silently splices unrelated shards into one series, and this is the most common way to get a wrong answer from the data.

One documented data correction is applied at load time by `Figure/near_data.py`: in the 8-shard era the workbook's `latency` sheet labels one shard `S2` while the throughput sheet, the aggregate sheet, and the workload-metric inputs label it `S4`, and the loader relabels it to `S4`. Throughput identifies `S4` as correct because it runs continuously across the 7-to-8-shard boundary whereas `S2` stops. The correction is itself a checked claim (E6) rather than only an assertion in prose. The workbook is shipped unmodified because it embeds 124 charts that would be lost if rewritten programmatically.

The aggregate correlations of E1 are computed across shards within a configuration, so n equals the shard count (4–9). Confidence intervals are correspondingly wide and are reported in full rather than summarized, and per-epoch significance tests at that sample size have limited power, as discussed in the paper. E2 holds per-shard gas and cross-shard ratio at their period averages, because the public extract carries no per-epoch values for them. It therefore tests stability under time-varying throughput, not under time-varying gas. Figures are regenerated rather than byte-compared, since matplotlib output varies with the fonts available on the host. Re-running the BigQuery extraction is optional, bills the reviewer's own Google Cloud project, and will return rows past the paper's cut-off, which is why the derived data is shipped.

The results are not sensitive to the exact pins. The full verification also passes under a deliberately different dependency set (`pandas` 2.2.3, `numpy` 2.5.1, `scipy` 1.18.0, `matplotlib` 3.11.1), spanning both current `pandas` major versions. If a check does fail, `verify.py` prints the observed value beside the expected one, and `--json` writes the same table in machine-readable form. We welcome contact through the submission system if any check does not reproduce.

References

[1] V. Buterin, "The meaning of decentralization," https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274, Feb 2017, accessed: December 15, 2025.

[2] ——, "A roadmap for scaling rollups with data sharding," https://notes.ethereum.org/@vbuterin/data_sharding_roadmap, 2021, accessed: December 31, 2025.

[3] V. Buterin *et al.*, "Ethereum white paper," *GitHub repository*, vol. 1, no. 22-23, pp. 5–7, 2013.

[4] C. Cachin, A. De Caro, P. Moreno-Sanchez, B. Tackmann, and M. Vukolic, "The transaction graph for modeling blockchain semantics," 2021.

[5] W. Chan and A. Olmsted, "Ethereum transaction graph analysis," in *2017 12th international conference for internet technology and secured transactions (ICITST)*. IEEE, 2017, pp. 498–500.

[6] J. Charlier, S. Lagraa, J. Francois *et al.*, "Profiling smart contracts interactions with tensor decomposition and graph mining," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery (ECML-PKDD)-Workshop on MIning DAta for financial applicationS (MIDAS)*, 2017.

[7] T. Chen, Z. Li, Y. Zhu, J. Chen, X. Luo, J. C.-S. Lui, X. Lin, and X. Zhang, "Understanding ethereum via graph analysis," *ACM Transactions on Internet Technology (TOIT)*, vol. 20, no. 2, pp. 1–32, 2020.

[8] W. Chen, T. Zhang, Z. Chen, Z. Zheng, and Y. Lu, "Traveling the token world: A graph analysis of ethereum erc20 token ecosystem," in *Proceedings of The Web Conference 2020*, 2020, pp. 1411–1421.

[9] Coindar. (2025, March) Near to hold hackathon on march 14th. Coindar event page. Accessed: March 17, 2026. [Online]. Available: https://coindar.org/en/event/near-to-hold-hackathon-on-march-14th-127545

[10] R. Dorfman, "A formula for the gini coefficient," *The review of economics and statistics*, pp. 146–149, 1979.

[11] D. Guo, J. Dong, and K. Wang, "Graph structure and statistical properties of ethereum transaction relationships," *Information Sciences*, vol. 492, pp. 58–71, 2019.

[12] H. Huang, X. Peng, J. Zhan, S. Zhang, Y. Lin, Z. Zheng, and S. Guo, "BrokerChain: A cross-shard blockchain protocol for account/balance-based state sharding," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1968–1977.

[13] Y. Huang, J. Tang, Q. Cong, R. T. Ma, L. Chen, and Y. M. Chee, "The last survivor of pos pools: Staker's dilemma," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 9, no. 1, pp. 1–29, 2025.

[14] L. Kiffer, D. Levin, and A. Mislove, "Analyzing ethereum's contract topology," in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 494–499.

[15] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598.

[16] D. Lin, J. Wu, Q. Yuan, and Z. Zheng, "Modeling and understanding ethereum transaction records via a complex network approach," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 11, pp. 2737–2741, 2020.

[17] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[18] NEAR Protocol, "Data Structures: Account," NEAR Protocol Specification (Nomicon), accessed: December 31, 2025. [Online]. Available: https://nomicon.io/DataStructures/Account.html

[19] ——. Near launches nightshade sharding, paving the way for mass adoption. Accessed: January 31, 2026. [Online]. Available: https://www.near.org/blog/near-launches-nightshade-sharding-paving-the-way-for-mass-adoption

[20] ——, "Near launches simple nightshade: The first step towards a sharded blockchain," Sep. 2021, accessed: November 4, 2025. [Online]. Available: https://pages.near.org/blog/near-launches-simple-nightshade-the-first-step-towards-a-sharded-blockchain/

[21] ——, "Near bigquery public dataset," 2025, accessed: November 11, 2025. [Online]. Available: https://docs.near.org/data-infrastructure/what-is#bigquery-public-dataset

[22] ——, "Near protocol: A scalable layer-1 blockchain," 2025, accessed: December 12, 2025. [Online]. Available: https://near.org/

[23] Y. Ren, J. Zhao, J. Li, and P. P. C. Lee, "An analysis of ethereum workloads from a key-value storage perspective," in *2025 IEEE International Symposium on Workload Characterization (IISWC)*, 2025.

[24] F. Saleh, "Blockchain without waste: Proof-of-stake," *The Review of financial studies*, vol. 34, no. 3, pp. 1156–1190, 2021.

[25] A. Skidanov and I. Polosukhin, "Nightshade: Near protocol sharding design," *URL: https://nearprotocol. com/downloads/Nightshade. pdf*, vol. 39, 2019.

[26] S. Somin, G. Gordon, and Y. Altshuler, "Network analysis of erc20 tokens trading on ethereum blockchain," in *International Conference on Complex Systems*. Springer, 2018, pp. 439–450.

[27] T. Z. Team, "The zilliqa technical whitepaper," https://docs.zilliqa.com/whitepaper.pdf, 2017.

[28] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2019, pp. 95–112.

[29] D. Yang, C. Long, H. Xu, and S. Peng, "A review on scalability of blockchain," in *Proceedings of the 2020 2nd International Conference on Blockchain Technology*, 2020, pp. 1–6.

[30] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 931–948.

[31] Y. Zhang, S. Pan, and J. Yu, "TxAllo: Dynamic transaction allocation in sharded blockchain systems," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 721–733.