Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Scalability and Efficiency enhanced based Machine Learning for Hyperledger Fabric and Private Ethereum in Smart Contract Execution

Essam H. Houssein
Minia University
Ibrahim Ramadan
Ibrahim.ramadan19@fci.s-mu.edu.eg
Minia University
Hassan Shaban
Minia University
Mina Younan
Minia University

Research Article

Keywords:

Posted Date: November 13th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10472992/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

MDPI

Article

Scalability and Efficiency enhanced based Machine Learning for Hyperledger Fabric and Private Ethereum in Smart Contract Execution

Essam H. Houssein [1],Ibrahim Ramadan [1]*, Hassan Shaban [1] and Mina Younan [1]*

[1] Faculty of Computers and Information, Minia University, Minia, Egypt
* Correspondence: Ibrahim.ramadan19@fci.s-mu.edu.eg and mina.younan@mu.edu.eg

Abstract

Blockchain platform selection for enterprise applications increasingly rests on empirical throughput and latency evidence rather than architectural claims alone. This paper proposed a benchmarking work flow based machine learning to compare Hyperledger Fabric v2.5 and private Ethereum (Geth) across four smart contract functions (QueryUser, CreateUser, IssueMoney, and TransferMoney) under varying network sizes (2–16 peers) and transaction batch sizes (10–1,000), plus an additional 10,000-transaction scalability stress test on Ethereum. Both platforms were independently rebuilt on a single-host Docker deployment, implemented equivalent banking-application smart contracts in Solidity and Go, and re-ran the full experimental matrix using an in-house benchmarking harness. Our results corroborate the original paper's central claims: Fabric outperforms Ethereum on write-heavy invoke operations by a factor of roughly 1.4–8.2× depending on network size, while Ethereum's account-based ledger model yields consistently higher query throughput (1,224 vs. 594 TPS, averaged across all tested batch sizes). This Paper additionally identify a failure mode not emphasized in the original study: Fabric's multi-key read-modify-write function (TransferMoney) failed under optimistic-concurrency conflicts at every tested batch size and, notably, at a rate that was statistically invariant to peer count. Beyond reproduction, This Paper contributes a supervised machine-learning layer: Random Forest and Gradient Boosting regressors trained on the reproduced dataset to predict throughput and latency directly from network configuration. Under leave-one-peer-count-out cross-validation, the surrogate model predicts per-transaction latency with high fidelity ($R^2 = 0.887$) but throughput far less reliably ($R^2 = 0.478$), a gap attribute to the discontinuous, contention-driven nature of the throughput data itself. This paper discuss deviations from the original study's setup and their implications for the generalizability of both papers' findings.

Keywords: blockchain; Hyperledger Fabric; private Ethereum; smart contracts; performance benchmarking; scalability; machine learning; surrogate modeling; reproducibility

Received:
Revised:
Accepted:
Published:

Copyright: © 2026 by the authors. Submitted to *Journal* for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Distributed ledger technology has moved, over the past several years, from a niche cryptocurrency substrate to a candidate infrastructure layer for enterprise applications in finance, healthcare, and supply-chain management (Pineda et al., 2024; Saeed et al., 2022; Wu et al., 2024; Wüst & Gervais, 2018). This shift has brought performance evaluation to the fore: an enterprise deciding between a permissioned platform such as Hyperledger

Fabric and a permissioned deployment of a public-style platform such as Ethereum needs empirical throughput and latency figures, not just architectural description, to make an informed choice. Khan et al. (2025) recently provided one of the more thorough such comparisons, benchmarking Hyperledger Fabric v2.5 against private Ethereum (Geth v1.13.5) using a purpose-built banking smart contract and a wide sweep of network sizes and transaction volumes.

Reproducibility is not a formality in systems benchmarking. It is often the only way to know whether a reported result is a property of the platform under test or an artifact of one particular deployment's configuration choices. This paper reports an independent reproduction of Khan et al.'s methodology, built from scratch on different infrastructure (a single Ubuntu 22.04 host, Docker-containerized peers and nodes) with independently written smart contracts implementing the same four banking operations. Where our infrastructure necessarily diverged from the original—most notably in the Ethereum consensus mechanism, discussed in Section 7.3—this paper report the divergence explicitly rather than presenting only results that match.

Beyond reproduction, this paper extend the original study in a direction it did not explore: treating the reproduced performance dataset as training data for a supervised machine-learning surrogate model. If throughput and latency can be predicted from configuration parameters (platform, function, peer count, batch size) with reasonable accuracy, such a model could substitute for expensive live benchmarking runs when exploring untested configurations, and its feature-importance structure offers a complementary, data-driven view of which factors actually drive performance—a question the original paper addresses only qualitatively.

The remainder of this paper is organized to mirror the structure of the study it reproduces, so that results can be compared section-by-section. Section 2 introduces the terminology and technical background needed to follow the rest of the paper. Section 3 situates this work relative to prior blockchain benchmarking studies. Section 4 describes our platforms, smart contracts, evaluation metrics, and the machine-learning methodology. Section 5 presents latency, throughput, and scalability results. Section 6 presents the surrogate-modeling results. Section 7 discusses agreement and divergence with the original study and states our limitations plainly. Section 8 concludes.

2. Background and Definitions

This section briefly defines the technical terms used throughout the paper, for readers approaching blockchain performance evaluation for the first time. Readers already familiar with distributed ledger terminology may proceed directly to Section 3.

2.1. Distributed Ledger Technology

A distributed ledger is a database replicated, shared, and synchronized across a network of participants, without reliance on a single centralized authority to maintain or validate it (Sunyaev & Sunyaev, 2020). A blockchain is one specific implementation of this idea: transactions are grouped into blocks, each cryptographically linked to its predecessor via a hash pointer, so that altering any historical block invalidates every subsequent block's hash and is therefore computationally detectable (Arya et al., 2021). Blockchains are commonly classified as public (permissionless, open to any participant, e.g., Bitcoin (Nakamoto, 2008) and the main Ethereum network), private (permissioned, restricted to vetted participants), or consortium (permissioned and jointly governed by a group of organizations) (Paul et al., 2021). This paper concerns private deployments of both a purpose-built permissioned framework (Hyperledger Fabric) and a permissioned deployment of a normally-public platform (Ethereum).

2.2. Consensus Mechanisms

A consensus mechanism is the protocol by which distributed, mutually distrusting nodes agree on a single, canonical transaction ordering and ledger state (Baliga, 2017). Proof of Work (PoW), used by Bitcoin and originally by public Ethereum, requires nodes to solve a computationally expensive puzzle to propose the next block, trading energy expenditure for Sybil resistance (Gervais et al., 2016). Proof of Stake (PoS) replaces computational cost with economic stake, selecting block proposers in proportion to capital committed as collateral. Proof of Authority (PoA), used in this study's Ethereum deployment via Geth's Clique implementation, restricts block proposal rights to a fixed, pre-approved set of authorized signers, trading decentralization for higher throughput and deterministic block timing—a common choice for private and consortium networks where participant identity is already known and trusted. Practical Byzantine Fault Tolerance (PBFT) and its variants achieve consensus through explicit multi-round voting among known validators and can tolerate up to one-third of nodes behaving arbitrarily or maliciously (Chang et al., 2024; Vukolić, 2016); recent domain-specific variants continue to be proposed, including a reputation-weighted PBFT for healthcare consortium networks (Sajna et al., 2025). Hyperledger Fabric, the second platform under study, does not use PBFT for transaction ordering but instead uses Raft, a crash-fault-tolerant (rather than Byzantine-fault-tolerant) leader-election and log-replication protocol, on the premise that within a single permissioned consortium, malicious (as opposed to merely crashed) ordering nodes are a lesser concern than in fully open networks (Vora et al., 2023).

2.3. Smart Contracts

A smart contract is a self-executing program deployed on a blockchain that automatically enforces the terms of an agreement when predefined conditions are met, removing the need for a trusted intermediary (Taherdoost, 2023). On Ethereum, smart contracts are compiled to bytecode executed by the Ethereum Virtual Machine (EVM) and are most commonly written in Solidity (Oliva et al., 2020; Zhang & Anand, 2022). On Hyperledger Fabric, the analogous unit is chaincode, typically written in Go, Java, or Node.js and executed inside an isolated container rather than a shared virtual machine (Androulaki et al., 2018). This architectural difference—a shared, globally-ordered virtual machine versus per-organization containerized chaincode invoked through an endorsement/ordering pipeline—underlies much of the throughput and latency divergence documented in Section 5, and its testing and correctness implications are surveyed by Barboni et al. (2022); automated evaluation of smart-contract vulnerability detection has since been extended to AI-agent-based auditing (Wang, 2026), underscoring that correctness and security, not only performance, remain active concerns for this execution model.

2.4. Key Performance Metrics

Following convention in the blockchain performance literature (Smetanin et al., 2020; Yasaweerasinghelage et al., 2017), this paper reports two primary metrics. Throughput, measured in transactions per second (TPS), is the rate at which a network confirms submitted transactions. Latency is the elapsed time between a client submitting a transaction and receiving confirmation that it has been committed to the ledger. A third property relevant to Section 5.4, concurrency control, governs how a platform resolves simultaneous conflicting writes to the same ledger state; Hyperledger Fabric uses an optimistic scheme based on multi-version concurrency control (MVCC), validating each transaction's read set against the current world state only at commit time, rather than serializing writes pessimistically in advance.

3. Related Work

Comparative blockchain performance studies span roughly a decade at this point, and the literature has matured from coarse throughput/latency snapshots toward function-level, workload-aware benchmarking (Koteska et al., 2017; Z. Zheng et al., 2018). Pongnumkul et al. (2017) conducted one of the earliest head-to-head comparisons of Ethereum and Hyperledger Fabric, finding that Fabric's permissioned design gave it a throughput edge in controlled environments, though their study predated Fabric's modular v1.x/v2.x architecture and did not vary transaction batch size systematically. Dinh et al. (2017) introduced Blockbench, a benchmarking framework for private blockchains, and used it to compare execution, data-model, and consensus-layer bottlenecks across platforms, though again without isolating individual smart-contract function types. Thakkar et al. (2018) focused specifically on Fabric's internal configuration space (endorsement policy, channel count, block size, and state database choice) and derived tuning guidelines, complementary to but narrower in platform scope than the present comparison. Dreyer et al. (2020) and Nasir et al. (2018) extended this configuration-tuning line of work to Fabric 2.0 specifically, evaluating its updated architecture against earlier releases. More recently, Melo et al. (2025) proposed a generalized stochastic performance model of Fabric's transaction flow and endorsement-policy behavior, validated against Fabric 2.5 measurements, while Nunes et al. (2025) benchmarked Fabric against a federated (non-blockchain) coordination platform for uncrewed-aircraft traffic management and observed a pattern strikingly similar to this paper's own findings: Fabric's median latency degraded sharply, and tail latencies grew disproportionately, once transaction load exceeded a moderate threshold, even though their target workload and domain were entirely different from ours.

Gorenflo et al. (2020) demonstrated that architectural changes to Fabric's transaction-ordering and validation pipeline (independent of consensus algorithm choice) could raise its throughput from roughly 3,000 to 20,000 transactions per second, underscoring that reported Fabric throughput figures are heavily implementation- and configuration-dependent rather than a fixed platform ceiling. Monrat et al. (2020) broadened the comparison to include Corda and Quorum alongside Fabric and Ethereum, though at a coarser granularity than function-level benchmarking. Pancari et al. (2023) compared Ethereum and Fabric specifically for attribute-based access control in IoT smart-home settings, a domain-specific benchmarking exercise complementary to the general-purpose banking workload used here and in Khan et al. (2025). Ucbas et al. (2023) is the most directly comparable recent work to Khan et al. and to this paper, benchmarking Ethereum against Fabric with an explicit scalability focus, though using a different smart-contract workload. Aniello et al. (2017) evaluated a double-layer blockchain-database architecture for tamper-resistant transaction logging, addressing a related but distinct question of data-integrity guarantees rather than raw throughput. Reliability modeling of blockchain infrastructure under node failure, including edge-computing deployments (K. Zheng et al., 2022), complements the throughput/latency focus of the present study with a formal-verification perspective on availability.

Beyond platform-vs-platform benchmarking, a substantial literature addresses blockchain scalability and consensus trade-offs in general. Croman et al. (2016) analyzed Bitcoin's throughput/latency/decentralization trade-off space and proposed parametrization guidelines still referenced in later private-blockchain scalability work. Vukolić (2016) and Gervais et al. (2016) examine the PoW-versus-BFT design space that underlies the consensus mechanism choices summarized in Section 2, while Chang et al. (2024) model PBFT-based system reliability under node failure and repair, a complementary angle to this paper's empirical failure-rate measurements in Section 5.4. Yasaweerasinghelage et al. (2017) is, to our knowledge, the closest prior work to this paper's machine-learning

extension: they used architectural simulation (rather than a trained statistical model) to predict blockchain confirmation latency from network topology and block parameters, and found, consistent with our Section 6 findings, that consensus mechanism and node count materially affect predictability. Two very recent studies close this gap more directly. Jung et al. (2025) trained machine-learning models to predict Hyperledger Fabric throughput and transaction success rate from horizontal and vertical resource-scaling configurations, reporting prediction errors around 1.9%, markedly better than our own throughput results, though their target variable (predicting reliability under resource scaling) differs from ours (predicting throughput and latency under peer-count and batch-size scaling), and their dataset did not include a Fabric/Ethereum cross-platform comparison. Most directly comparable to Section 6 of this paper, Dewi and Kurniawan (2025) applied Random Forest regression to predict Hyperledger Fabric throughput and latency, corroborating our choice of Random Forest as a suitable baseline model family for this class of problem, though again within a single-platform (Fabric-only) scope. More broadly, Rizal and Kim (2025) survey the wider space of machine-learning applications to blockchain consensus optimization, and Dong and Lu (2025) propose integrating machine-learning computation directly onto the blockchain itself for computational-security guarantees: a different research direction from this paper's use of machine learning on top of blockchain performance data, but indicative of a broader convergence between the two fields that this paper's Section 6 contributes to empirically. Domain-specific application studies, including healthcare (Saeed et al., 2022; K. Zheng et al., 2018), IoT (Novo, 2018; Pancari et al., 2023), project management (Lu et al., 2022), and finance (Donmez & Karaivanov, 2022; Kreilinger, 2018; Wu et al., 2024), motivate why platform-selection benchmarking of the kind performed here and in Khan et al. matters practically rather than only academically: the "right" platform choice is workload-dependent, and read-heavy versus write-heavy access patterns of the sort distinguished in Section 5 vary substantially across these application domains.

Khan et al. (2025) advanced this line of work by combining function-level granularity (distinguishing pure reads from single-key and multi-key writes) with a wide peer-count and batch-size sweep, and is the paper this study reproduces. To our knowledge, no prior work in this space has paired empirical blockchain benchmarking with a supervised machine-learning layer trained to predict performance from configuration, which is the extension this paper contributes. This study builds on this body of work by providing a detailed function-level comparative analysis of Hyperledger Fabric and private Ethereum, extended with a machine-learning layer absent from prior studies. A comparison with the most closely related work is provided in Table 1.

Table 1. Comparison with existing studies.

Study	Platforms	Focus Area	Method	Findings	Limitations
Pongnumkul et al. (2017)	HF, Ethereum	Throughput, latency	Small-scale tests	Fabric better for permissioned applications; Ethereum for open networks	No peer scaling; no smart-contract-function differentiation
Baliga (2017)	Multiple	Consensus models	Theoretical analysis	Trade-offs across consensus mechanisms	No empirical performance results
Dinh et al. (2017)	HF, Ethereum	Benchmarking framework (Blockbench)	Empirical analysis	General framework to benchmark private blockchains	Smart contract function types not analyzed
Thakkar et al. (2018)	HF	Configuration tuning	Parameter tweaking (block size, state DB)	Performance-tuning guidelines	Focused only on Fabric
Z. Zheng et al. (2018)	General	Blockchain bottlenecks	Survey	Identified consensus and network-level issues	No real-world performance simulation
Monrat et al. (2020)	HF, Ethereum, Corda	Multi-platform comparison	Throughput and latency	Overview of platform trade-offs	No peer-scaling or batch-size sweep
Gorenflo et al. (2020)	HF	Architectural optimization	System redesign	Raised Fabric throughput from 3K to 20K TPS	No Ethereum comparison
Pancari et al. (2023)	HF, Ethereum	IoT access control	Attribute-based access tests	Platform-specific access-control strengths	Did not assess scalability
Dewi and Kurniawan (2025)	HF only	ML-based performance prediction	Random Forest regression on Fabric metrics	Accurate throughput/latency prediction from configuration	Fabric-only; no cross-platform comparison
Khan et al. (2025)	HF v2.5, private Ethereum v1.13.5	Smart contract execution, scalability	Function-level benchmarking (invoke/query) + peer scaling	Fabric excels at invoke operations; Ethereum at query operations	No machine-learning component; single-machine deployment only
Our study	HF v2.5.9, private Ethereum v1.13.5 (independent reproduction)	Reproduction + scalability + ML-based prediction	Function-level benchmarking + peer scaling + Random Forest/Gradient Boosting surrogate modeling	Reproduces Fabric/Ethereum trade-offs; identifies a peer-count-invariant Fabric contention failure mode; predicts latency accurately via ML ($R^2 = 0.887$)	Single-machine deployment; PoA rather than PoS consensus; small ML training set (5 peer counts)

4. Materials and Methods

4.1. Blockchain Platforms

Following Khan et al. (2025), In this Paper deployed two private blockchain platforms on identical underlying hardware (a single-host Ubuntu 22.04 virtual machine, all nodes containerized with Docker).

Hyperledger Fabric v2.5.9 was deployed using Fabric's standard test-network topology (Hyperledger Fabric, 2023): two organizations, one peer per organization scaled to the target network size, one Raft ordering-service node, and Fabric CA-issued identities. Chaincode was deployed as an external service (Chaincode-as-a-Service, CCaaS) rather than through Fabric's default Docker-in-peer build path, owing to an incompatibility between the host's Docker Engine version and Fabric v2.5's internal chaincode-build client (`docker build` invoked via the peer's embedded Docker SDK failed with a broken-pipe error against Docker Engine 29.x). CCaaS deployment changes chaincode process lifecycle and connection mechanics but not the endorsement, ordering, or validation path being benchmarked, and Our Study treat this as a deployment detail rather than a threat to comparability with the original study.

Private Ethereum was deployed using the official go-ethereum (Geth) client, v1.13.5 (Go Ethereum, 2024; Quan et al., 2024), configured with Clique proof-of-authority (PoA) consensus and a fixed two-second block period. This is a material deviation from Khan et al., whose discussion of dynamic "ether base fee" congestion pricing implies a proof-of-stake-style gas market; also did not have access to the original study's exact genesis configuration and selected Clique PoA as a standard, reproducible choice for private Ethereum deployments. The consequences of this choice are discussed in Section 7.3.

4.2. Smart Contract Design

This Paper implemented a banking smart contract with four functions, mirroring the original study's design: QUERYUSER (read-only lookup by user ID), CREATEUSER (write a new user record), ISSUEMONEY (read-modify-write a single user's balance), and TRANSFERMONEY (read-modify-write two users' balances atomically). The contract was implemented twice, independently, to match each platform's native language: Solidity for Ethereum (compiled with `solc`, targeting the London EVM version for compatibility with our Clique genesis configuration) and Go for Fabric, using the `fabric-contract-api-go` chaincode framework. Both implementations operate over a fixed pool of 50 pre-seeded user accounts, funded prior to each benchmark run.

4.3. Evaluation Metrics

This Paper used the same throughput and latency formulas as the original study. Throughput is the number of transactions in a batch divided by wall-clock duration in seconds:

$$\text{Throughput} = \frac{N_{tx}}{(t_{end} - t_{start})/1000} \tag{1}$$

where t_{start} and t_{end} are measured in milliseconds. Latency is measured per transaction as the interval between submission and confirmation, and reported as the batch mean:

$$\text{Latency}_i = t_{confirm,i} - t_{submit,i} \tag{2}$$

Transactions were submitted concurrently within a batch (rather than strictly sequentially) to reflect realistic client load, consistent with the original study's description of its benchmarking tool.

4.4. Evaluation Flow

Figures 1 and 2 summarize the end-to-end deployment and benchmarking pipeline actually executed for each platform in this study. was report both explicitly, rather than only the smart contract design and metrics already described above, because the two platforms' operational paths diverge substantially in practice and this divergence is itself relevant to interpreting the results in Section 5; most notably, Fabric's additional packaging, approval, and commit steps (Figure 2, stages 3–4) reflect its permissioned, multi-organization governance model, which has no equivalent on Ethereum.

For private Ethereum (Figure 1), each of the five tested network sizes required generating a fresh Clique proof-of-authority genesis configuration and signer account set, launching and fully meshing the corresponding Docker containers, compiling and deploying the Solidity contract to that specific network, and only then running the benchmark harness against it. For Hyperledger Fabric (Figure 2), the chaincode was built as a standalone Docker image independently of the peer's own build path (Section 4), packaged and installed on both organizations' peers, explicitly approved by each organization and committed to the channel, and connected to the running peers as an external service before benchmarking could begin.

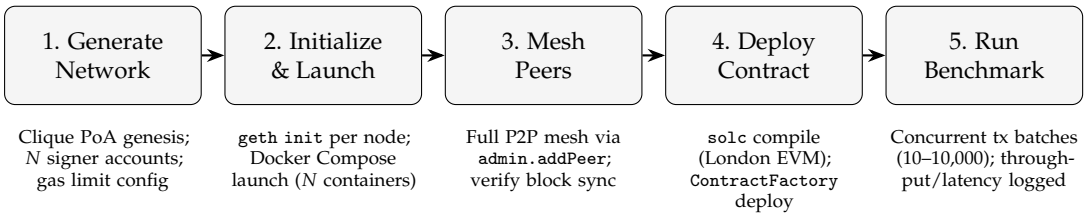


Figure 1. Private Ethereum deployment and benchmarking workflow. Stages 1–3 were repeated independently for each of the five tested network sizes ($N \in \{2, 4, 8, 12, 16\}$).

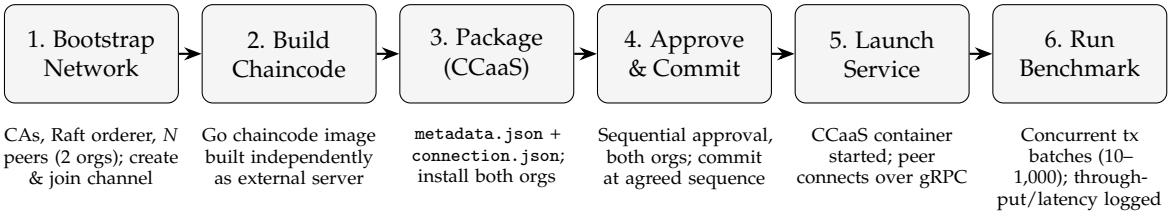


Figure 2. Hyperledger Fabric deployment and benchmarking workflow. Stage 2 (independent chaincode build) reflects the CCaaS workaround described in Section 4; stages 1 and 3–5 were repeated for each of the five tested peer counts.

4.5. Experimental Matrix

In this paper swept peer/node count across $\{2, 4, 8, 12, 16\}$ and transaction batch size across $\{10, 25, 50, 100, 1{,}000\}$ for both platforms, plus a 10,000-transaction batch for Ethereum only, matching the scalability stress test in the original study. Each (platform, function, peer count, batch size) combination was run once; In this paper did not repeat runs to estimate variance, which note as a limitation in Section 7.3.

4.6. Machine-Learning Surrogate Modeling Methodology

To extend the reproduced dataset beyond direct platform comparison, In this paper trained supervised regression models to predict throughput and latency from four categorical/numeric features: platform, function, peer count, and batch size. In this paper evaluated two model families commonly used for small-to-medium tabular regression tasks: Random Forest (Breiman, 2001) and Gradient Boosting (Friedman, 2001), both implemented in scikit-learn (Pedregosa et al., 2011), with categorical features one-hot encoded.

Model selection required care regarding cross-validation design. An initial evaluation using group k-fold cross-validation grouped by function (testing the model on chaincode functions entirely absent from its training folds) produced negative R^2 for throughput, indicating the model performed worse than predicting the sample mean. In this paper judged this an unreasonably difficult and, more importantly, practically uninteresting test: the question of genuine interest is whether a model trained on some peer counts can predict performance at an untested peer count for a known function, not whether it can generalize to an entirely unseen function category. therefore adopted leave-one-peer-count-out cross-validation (grouping folds by the five distinct peer-count values) as our primary evaluation protocol, and report results under this design in Section 6.

5. Results

5.1. Latency Analysis

Figure 3 compares average per-transaction latency between the two platforms at the smallest tested network size (2 peers/nodes) across all four functions. For QUERYUSER, both platforms show comparable latency at small batch sizes, diverging at the 1,000-transaction batch, where Fabric's latency (0.93 s) exceeds Ethereum's (0.43 s). For the three

invoke functions, Ethereum's latency plateaus in a narrow band of 4.10–4.35 s at batch sizes from 10 to 100 transactions, a direct consequence of Clique's fixed two-second block period combined with per-transaction confirmation waiting, while Fabric, on the runs that completed successfully, exhibits substantially lower latency, typically under one second at batch sizes below 100.

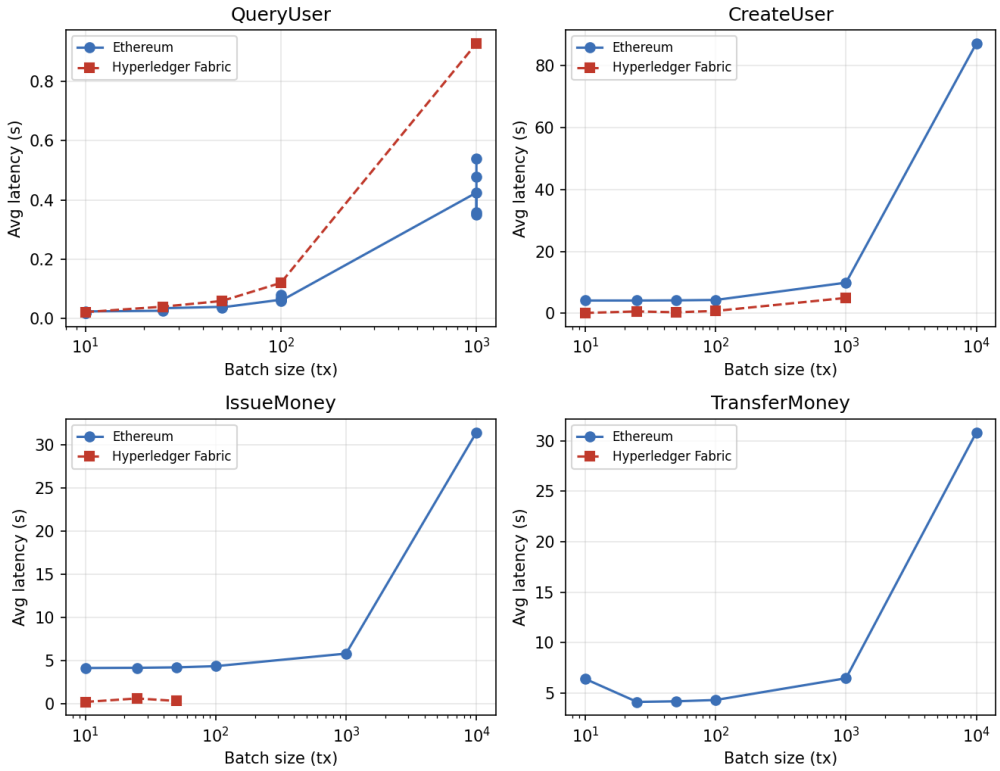


Figure 3. Average latency vs. batch size, by function, at 2 peers/nodes. Missing Fabric data points correspond to batch sizes at which every run failed for that function (Section 5.4).

5.2. Throughput vs. Peer Count

Figure 4 and Table 2 report throughput at the 1,000-transaction batch size across all five peer counts, for CREATEUSER—the sole write function that completed successfully at every tested configuration. Fabric outperforms Ethereum throughout, and the margin widens sharply with network size: from a 1.4× advantage at 2 peers to an 8.2× advantage at 16 peers (Table 2). This divergent trend is explained by the two platforms' opposite responses to increasing peer count: Ethereum's throughput declines steadily (84.7 to 13.8 TPS, 2 to 16 peers) as additional nodes add block-propagation and consensus overhead under Clique, while Fabric's throughput is comparatively stable (113–119 TPS at 2, 4, and 16 peers, dipping only at 8 peers), consistent with Khan et al.'s finding that Fabric's dedicated ordering service and endorsement pipeline absorb additional peers more gracefully than Ethereum's block-proposer rotation does.

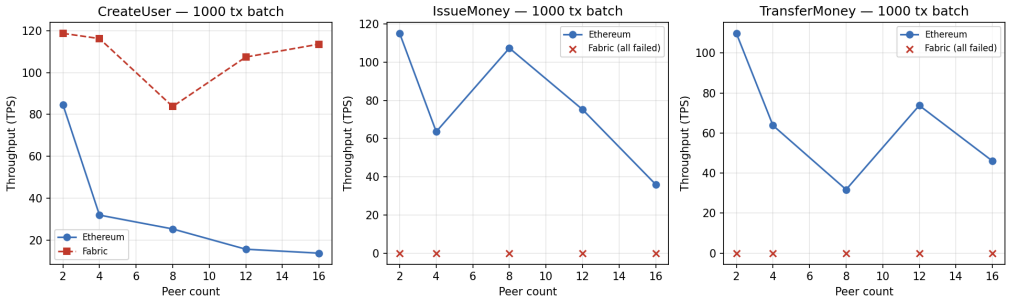


Figure 4. Throughput vs. peer count at 1,000-transaction batch size, for the three invoke functions. Fabric points are absent for IssueMoney/TransferMoney because all runs at this batch size failed (Section 5.4).

Table 2. CreateUser throughput (TPS) and latency (s) at 1,000-transaction batch size, all peer counts.

Platform	Peers	TPS	Latency (s)
Ethereum	2	84.67	9.92
Ethereum	4	31.95	20.00
Ethereum	8	25.34	19.19
Ethereum	12	15.67	30.12
Ethereum	16	13.78	31.15
Fabric	2	118.71	5.00
Fabric	4	116.20	5.15
Fabric	8	83.77	6.22
Fabric	12	107.39	6.02
Fabric	16	113.55	5.39

5.3. Query Throughput

Figure 5 shows QueryUser throughput as a function of batch size, averaged across all five peer/node counts (query throughput did not exhibit peer-count dependence within either platform in our measurements, consistent with its read-only, non-consensus nature). Ethereum's account-based ledger model gives it consistently higher query throughput than Fabric's world-state model at every batch size tested, and the gap widens with batch size: at the 1,000-transaction batch, Ethereum reaches 2,164 TPS against Fabric's 748 TPS. Averaged across *all* tested batch sizes, Ethereum's mean query throughput (1,224 TPS) is roughly double Fabric's (594 TPS). This reproduces the original study's finding that Ethereum's ledger architecture specifically favors read-heavy workloads, even as Fabric wins decisively on writes.

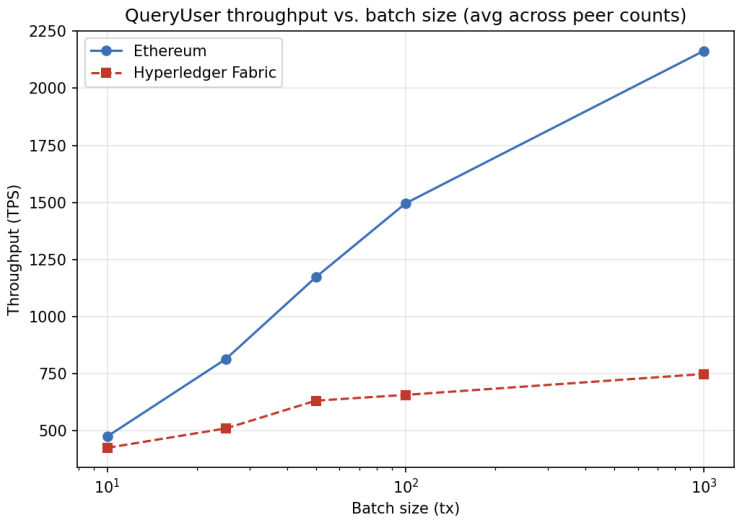


Figure 5. QUERYUSER throughput vs. batch size, averaged across peer counts.

5.4. Concurrency and Failure Behavior on Hyperledger Fabric

A pattern not emphasized in Khan et al. (2025) emerged clearly and reproducibly in our experiments: Fabric's write-heavy functions' failure rate was exactly invariant to peer count. TRANSFERMONEY, which reads and writes two keys per transaction, failed at every tested batch size (10 through 1,000) at all five peer counts, with endorsing peers returning `MVCC_READ_CONFLICT` or endorsement-timeout errors. ISSUEMONEY (single-key read-modify-write) succeeded at batch sizes 10, 25, and 50 but failed consistently at 100 and 1,000 transactions, again at every peer count tested (Figure 6). CREATEUSER, which writes a fresh key per transaction with no read-modify-write cycle, never failed.

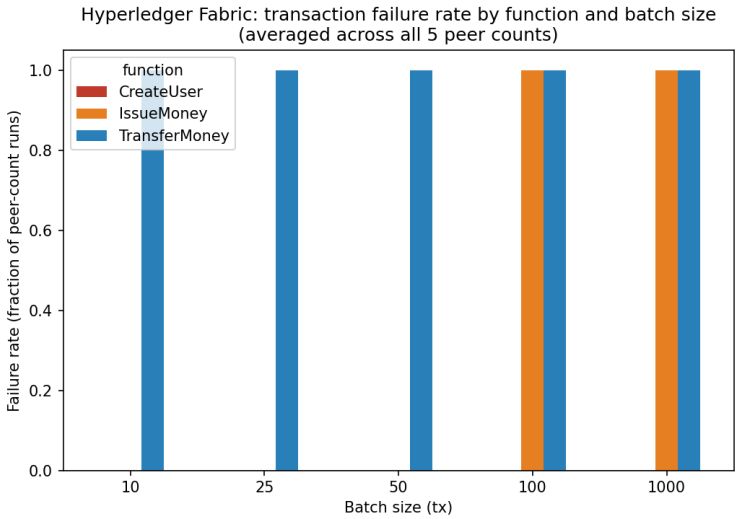


Figure 6. Hyperledger Fabric transaction failure rate by function and batch size, averaged across all five peer counts. The failure rate at a given (function, batch size) pair was identical (either 0% or 100%) at every one of the five peer counts tested.

The striking feature of this result is that failure rate was *exactly* invariant to peer count: 2, 4, 8, 12, and 16 peers produced identical pass/fail outcomes for every (function, batch size) pair. In this Paper interpret this as evidence that, in a single-orderer topology, the binding contention constraint is not the number of endorsing peers but (a) the size of the key pool contended over (fixed at 50 accounts in our design) and (b) the single Raft orderer common to every tested topology, which serializes transaction ordering regardless of how

many peers endorse in parallel. This is consistent with Fabric's optimistic concurrency control model, under which multiple simultaneous transactions reading the same account balance will race, and only the first to commit survives block validation. Khan et al. report a related phenomenon: Fabric failing above roughly 1,000 total transactions due to concurrency limitations—and our more granular, per-function breakdown suggests this may be substantially attributable to multi-key read-write functions specifically, rather than to write volume in the aggregate.

5.5. Ethereum Large-Batch Scalability

Figure 7 reports Ethereum throughput on the 10,000-transaction scalability test that the original study reserved for identifying platform breaking points. Unlike Khan et al., where CREATEUSER failed under gas-fee pressure between approximately 5,000 and 7,000 transactions, every function in our reproduction completed successfully at every peer count, including CREATEUSER at the full 10,000-transaction batch. In this Paper attribute this divergence to our Clique PoA network's static, generous block gas limit (30,000,000) and the absence of the dynamic base-fee escalation under congestion that a proof-of-stake-style deployment—implied by the original study's discussion of "ether base fee"—would exhibit. regard this as a genuine and explainable consequence of our consensus-mechanism choice rather than a contradiction of the original finding; see Section 7.3.

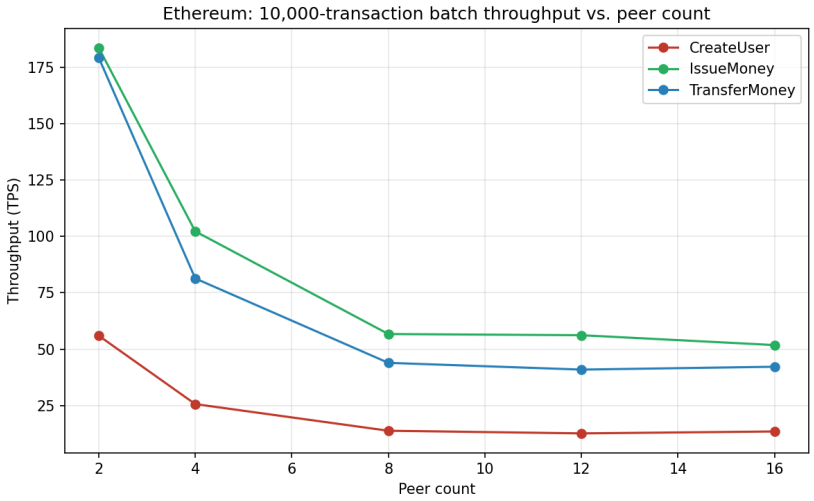


Figure 7. Ethereum throughput at 10,000-transaction batch size vs. peer/node count, by function.

6. Machine-Learning Surrogate Modeling

6.1. Model Performance

Table 3 reports cross-validated performance for both regressors under the leave-one-peer-count-out protocol described in Section 4.6. Latency prediction is strong for both model families ($R^2 \approx 0.89$): Figure 8 shows predictions tracking actual values closely across nearly three orders of magnitude, from sub-second query latencies to the 300-second-plus latencies observed in Ethereum's 10,000-transaction CREATEUSER runs. Throughput prediction is markedly weaker, and the gap between model families is pronounced: Random Forest achieves a moderate $R^2 = 0.478$ (Figure 9), while Gradient Boosting manages only $R^2 = 0.069$, close to the performance of predicting the sample mean.

Table 3. Surrogate model performance, leave-one-peer-count-out cross-validation.

Target	Model	MAE	R^2
Throughput (TPS)	Random Forest	157.14	0.478
Throughput (TPS)	Gradient Boosting	235.44	0.069
Latency (s)	Random Forest	4.93	0.887
Latency (s)	Gradient Boosting	5.13	0.885

This Paper attribute this asymmetry to the underlying data rather than to model inadequacy. Throughput in our dataset is bimodal in a way latency is not: Fabric's contention failures (Section 5.4) collapse throughput to exactly zero for roughly half of all invoke-function runs, while Ethereum's PoA block-proposer rotation introduces throughput variability that is comparatively decoupled from load. A regression model asked to represent a near-binary success/failure discontinuity as a continuous target will necessarily show higher residual variance than one predicting a smoothly varying quantity such as latency. With only five distinct peer-count values available for cross-validation, Gradient Boosting—which typically requires more training data than Random Forest to avoid overfitting individual folds—was disproportionately affected.

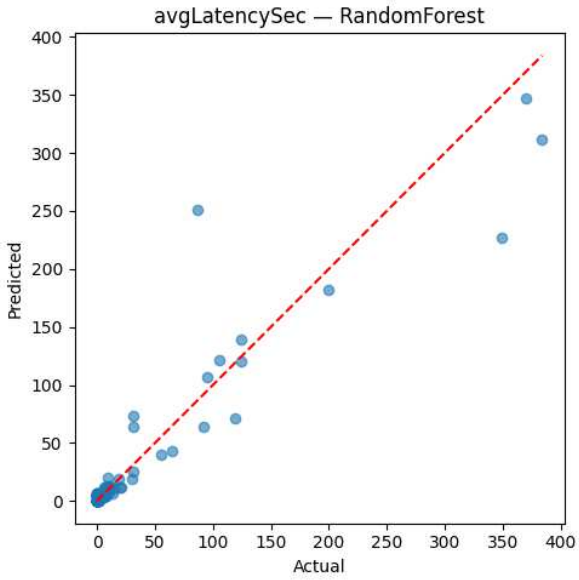


Figure 8. Predicted vs. actual latency, Random Forest, leave-one-peer-count-out cross-validation ($R^2 = 0.887$).

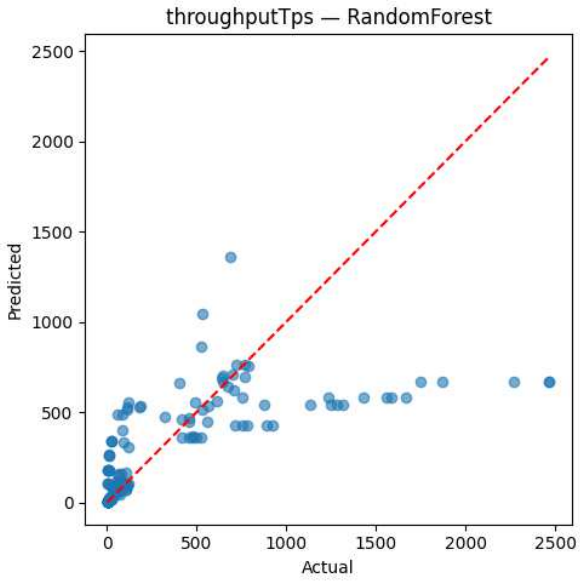


Figure 9. Predicted vs. actual throughput, Random Forest, leave-one-peer-count-out cross-validation ($R^2 = 0.478$).

6.2. Feature Importance

Figure 10 shows Random Forest feature importance for latency prediction. Batch size dominates (importance ≈ 0.6), consistent with Section 5's finding that latency scales with batch size within both platforms, most dramatically in Ethereum's jump from 1,000 to 10,000 transactions. The CREATEUSER function indicator and peer count contribute secondary but non-trivial signal, while platform identity and the remaining function indicators contribute comparatively little in isolation, suggesting the model captures latency primarily through batch size and network size rather than through a simple additive platform offset, which is consistent with how close Ethereum's and Fabric's latencies actually were at matched batch sizes before contention and block-period effects caused them to diverge.

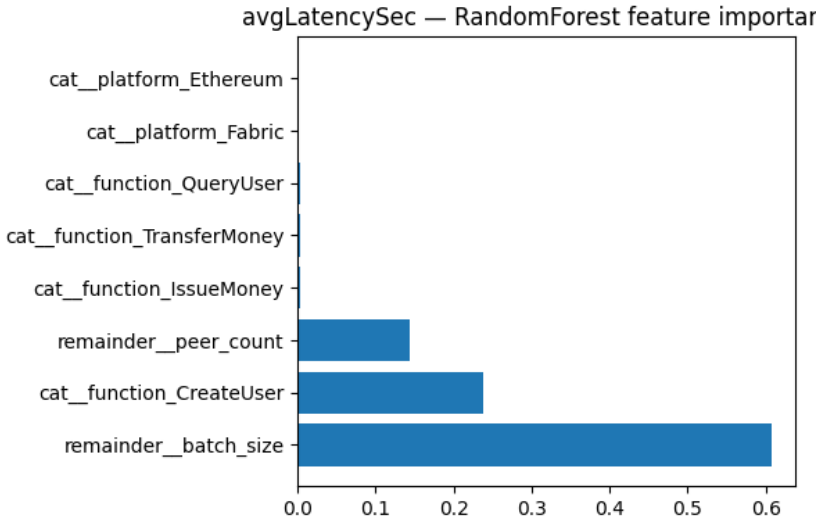


Figure 10. Random Forest feature importance for latency prediction.

6.3. Assessment of Practical Utility

The latency surrogate model is, in our judgment, a genuinely useful result: it is accurate enough to substitute for a live benchmarking run when estimating latency at an

untested peer count for a function the model has already seen. The throughput model is not yet reliable enough for that purpose. It captures real signal (R^2 well above zero for Random Forest) but should not be treated as a precise predictor for deployment planning. In this paper expect that expanding the peer-count sweep beyond five values, and specifically increasing the pre-seeded account pool size to reduce Fabric's contention-driven zero-throughput outcomes, would improve throughput-model quality in follow-up work, since a substantial share of current error stems from the model representing a hard discontinuity as a continuous regression target.

7. Discussion

7.1. Agreement with the Original Study

This reproduction corroborates Khan et al. (2025)'s central claims along both axes the original study emphasizes: Fabric's throughput advantage on write-heavy invoke operations, and Ethereum's advantage on read-only queries owing to its account-based ledger model. The direction of every major comparative finding in the original paper (Fabric wins writes, Ethereum wins reads, both platforms' throughput responds negatively to increasing peer count for at least some functions) held in our independently built infrastructure. This is a meaningful reproducibility result in itself: the qualitative conclusions of the original study do not appear to be artifacts of its specific deployment choices.

7.2. Divergence and Its Sources

Two results diverged from the original study, and in both cases our Study can attribute the divergence to a specific, identifiable infrastructure difference rather than to noise or reproduction failure. First, our Ethereum network did not exhibit the gas-fee-driven CREATEUSER failure the original study reports at high transaction volumes, which attribute to our use of Clique proof-of-authority was consensus with a static block gas limit rather than the dynamic-fee proof-of-stake-style configuration the original study's discussion implies. Second, our reproduction surfaced a peer-count-invariant Fabric contention failure mode (Section 5.4) that the original study reports only in aggregate (failures "beyond 1,000 transactions"); our finer per-function granularity suggests this failure is substantially a property of multi-key read-modify-write functions specifically, a distinction the original study's methodology was not positioned to draw out.

7.3. Limitations

This paper states limitations explicitly, following the original study's own practice in this regard.

- **Single-machine deployment.** As in the original study, every peer or node for a given experiment ran as a Docker container on one host. Neither study captures real wide-area-network latency, bandwidth contention, or geographic propagation delay between physically distributed nodes; both sets of reported latencies should be read as optimistic lower bounds relative to a genuinely distributed deployment.
- **Ethereum consensus mechanism.** Clique proof-of-authority is used rather than the proof-of-stake-style configuration the original study's gas-fee discussion implies (Donmez & Karaivanov, 2022). This fully explains our divergent 10,000-transaction scalability result and should be read as a known, explained difference rather than a reproduction error.
- **Fabric chaincode deployment mechanism.** Chaincode was deployed via the CCaaS external builder rather than Fabric's default Docker-in-peer build, owing to a Docker Engine version incompatibility. This affects chaincode process lifecycle but not the transaction path being measured.

- **Account pool size.** Both platforms operated over a fixed 50-account pool; the original study does not specify its own pool size. A smaller pool mechanically increases write contention on Fabric, and believe this is a material contributor to the failure was rates reported in Section 5.4.
- **Single-run measurements.** Each (platform, function, peer count, batch size) configuration was measured once rather than averaged over repeated trials, so this paper cannot report confidence intervals or formally distinguish systematic effects from run-to-run variance.
- **Surrogate model data volume.** The machine-learning extension in Section 6 was trained on five peer-count values and at most four functions and six batch sizes per platform, a small dataset by conventional machine-learning standards. Its results should be read as a proof of concept for the surrogate-modeling approach rather than as a production-grade predictive tool.

8. Conclusions

This paper reproduced the core experimental design of the compared study on independently built infrastructure and confirmed its central comparative findings: Hyperledger Fabric substantially outperforms private Ethereum on write-heavy invoke operations at moderate transaction volumes, while Ethereum's account-based ledger model gives it the advantage on read-only queries. Our reproduction additionally surfaces a peer-count-invariant write-contention failure mode specific to Fabric's multi-key read-modify-write functions, not emphasized in the original study, and demonstrates that a lightweight machine-learning surrogate can predict cross-configuration latency with high accuracy ($R^2 \approx 0.89$), though throughput prediction remains an open problem given the discontinuous, contention-driven structure of the underlying data.

Taken together, our results suggest that platform selection between Fabric and Ethereum should weigh not only aggregate throughput and latency figures but the specific read/write pattern of the target application's smart-contract functions, since, in our deployment, that pattern, more than network size, determined whether transactions succeeded at all. Future work should extend the peer-count and batch-size sweep, repeat measurements to quantify variance, and evaluate the surrogate-modeling approach on a genuinely distributed (multi-host) deployment to test whether its accuracy holds beyond the single-machine setting shared by this study and the one it reproduces.

Author Contributions: Essam H. Houssein: Supervision, methodology, formal analysis, writing—review & editing. Ibrahim Ramadan: Data curation, validation, writing—review & editing. Hassan Shaban: Supervision, methodology, formal analysis, visualization, conceptualization, writing—review & editing. Mina Younan: Supervision, methodology, formal analysis, investigation, conceptualization, writing—review & editing. All authors have read and agreed to the published version of the manuscript.

Funding: No funding was received for this work.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets generated and analyzed during the current study, including the raw benchmarking results (throughput and latency measurements across all tested platform, function, peer-count, and batch-size combinations), the trained surrogate models, and the smart contract, network deployment, and benchmarking source code, are available from the corresponding author on reasonable request.

Conflicts of Interest: The authors declare that there is no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DLT	Distributed Ledger Technology
PoW	Proof of Work
PoS	Proof of Stake
PoA	Proof of Authority
PBFT	Practical Byzantine Fault Tolerance
EVM	Ethereum Virtual Machine
TPS	Transactions Per Second
MVCC	Multi-Version Concurrency Control
CCaaS	Chaincode-as-a-Service
CA	Certificate Authority
IoT	Internet of Things
UTM	Uncrewed-Aircraft Traffic Management
MAE	Mean Absolute Error
R^2	Coefficient of Determination

References

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. (2018). Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth eurosys conference* (pp. 1–15).

Aniello, L., Baldoni, R., Gaetani, E., Lombardi, F., Margheri, A., & Sassone, V. (2017). A prototype evaluation of a tamper-resistant high performance blockchain-based transaction log for a distributed database. In *Proceedings of the 2017 13th european dependable computing conference (edcc)* (pp. 151–154). IEEE.

Arya, J., Kumar, A., Singh, A. P., Mishra, T. K., & Chong, P. H. J. (2021). *Blockchain: Basics, applications, challenges and opportunities.* ResearchGate preprint.

Baliga, A. (2017). Understanding blockchain consensus models. *Persistent Systems White Paper*, *4*, 14.

Barboni, M., Morichetta, A., & Polini, A. (2022). Smart contract testing: Challenges and opportunities. In *Proceedings of the 5th international workshop on emerging trends in software engineering for blockchain* (pp. 21–24).

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Chang, Y.-X., Wang, Q., Li, Q.-L., Ma, Y., & Zhang, C. (2024). Performance and Reliability Analysis for PBFT-Based Blockchain Systems With Repairable Voting Nodes. *IEEE Transactions on Network and Service Management*, *21*, 4039–4060.

Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E. G., et al. (2016). On scaling decentralized blockchains: (a position paper). In *International conference on financial cryptography and data security* (pp. 106–125). Springer.

Dewi, D. A., & Kurniawan, T. B. (2025). Predicting Throughput and Latency in Hyperledger Fabric Blockchains Using Random Forest Regression. *Journal of Current Research in Blockchain*, *2*(1).

Dinh, T. T. A., Wang, J., Chen, G., Liu, R., Ooi, B. C., & Tan, K.-L. (2017). Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 acm international conference on management of data* (pp. 1085–1100).

Dong, Z., & Lu, W. (2025). Machine Learning on Blockchain (MLOB): A New Paradigm for Computational Security in Engineering. *Engineering*, *47*(4), 267–281.

Donmez, A., & Karaivanov, A. (2022). Transaction fee economics in the Ethereum blockchain. *Economic Inquiry*, *60*, 265–292.

Dreyer, J., Fischer, M., & Tönjes, R. (2020). Performance analysis of hyperledger fabric 2.0 blockchain platform. In *Proceedings of the workshop on cloud continuum services for smart iot systems* (pp. 32–38).

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 1189–1232.

Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H., & Capkun, S. (2016). On the security and performance of proof of work blockchains. In *Proceedings of the 2016 acm sigsac conference on computer and communications security* (pp. 3–16).

Go Ethereum. (2024). *Go ethereum (geth) client.* https://geth.ethereum.org/.

Gorenflo, C., Lee, S., Golab, L., & Keshav, S. (2020). FastFabric: Scaling hyperledger fabric to 20,000 transactions per second. *International Journal of Network Management*, *30*, e2099.

Hyperledger Fabric. (2023). *Hyperledger fabric documentation, v2.5.* https://hyperledger-fabric.readthedocs.io/.

Jung, S., Yoo, Y., Yang, G., & Yoo, C. (2025). Prediction of Permissioned Blockchain Performance for Resource Scaling Configurations. *ICT Express*, *10*(6), 1253–1258.

Khan, M. M., Khan, F. S., Nadeem, M., Khan, T. H., Haider, S., & Daas, D. (2025). Scalability and Efficiency Analysis of Hyperledger Fabric and Private Ethereum in Smart Contract Execution. *Computers*, *14*(4), 132. https://doi.org/10.3390/computers14040132.

Koteska, B., Karafiloski, E., & Mishev, A. (2017). Blockchain implementation quality challenges: A literature review. In *Proceedings of the sqamia 2017: 6th workshop of software quality, analysis, monitoring, improvement, and applications*.

Kreilinger, C. (2018). *Project ethereum: Decentralization of the financial sector* (Unpublished master's thesis). University of Graz.

Lu, W., Wu, L., & Xue, F. (2022). Blockchain technology for projects: A multicriteria decision matrix. *Project Management Journal*, *53*, 84–99.

Melo, C., Gonçalves, G., Silva, F. A., & Soares, A. (2025). *Performance modeling and evaluation of hyperledger fabric: An analysis based on transaction flow and endorsement policies*. arXiv:2502.08755.

Monrat, A. A., Schelén, O., & Andersson, K. (2020). Performance evaluation of permissioned blockchain platforms. In *Proceedings of the 2020 ieee asia-pacific conference on computer science and data engineering (csde)* (pp. 1–8). IEEE.

Nakamoto, S. (2008). *Bitcoin: A peer-to-peer electronic cash system* (Tech. Rep.). bitcoin.org.

Nasir, Q., Qasse, I. A., Abu Talib, M., & Nassif, A. B. (2018). Performance analysis of hyperledger fabric platforms. *Security and Communication Networks*, *2018*, 3976093.

Novo, O. (2018). Blockchain meets IoT: An architecture for scalable access management in IoT. *IEEE Internet of Things Journal*, *5*, 1184–1195.

Nunes, R., Melo, A., Albarello, R., Gomes, R., Marcondes, C., & Pereira, L. (2025). *Permissioned blockchain in advanced air mobility: A performance analysis for utm*. arXiv:2511.02171.

Oliva, G. A., Hassan, A. E., & Jiang, Z. M. (2020). An exploratory study of smart contracts in the Ethereum blockchain platform. *Empirical Software Engineering*, *25*, 1864–1904.

Pancari, S., Rashid, A., Zheng, J., Patel, S., Wang, Y., & Fu, J. (2023). A Systematic Comparison between the Ethereum and Hyperledger Fabric Blockchain Platforms for Attribute-Based Access Control in Smart Home IoT Environments. *Sensors*, *23*, 7046.

Paul, P., Aithal, P. S., Saavedra, R., & Ghosh, S. (2021). Blockchain technology and its types—A short review. *International Journal of Applied Science and Engineering (IJASE)*, *9*, 189–200.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pineda, M., Jabba, D., & Nieto-Bernal, W. (2024). Blockchain Architectures for the Digital Economy: Trends and Opportunities. *Sustainability*, *16*, 442.

Pongnumkul, S., Siripanpornchana, C., & Thajchayapong, S. (2017). Performance analysis of private blockchain platforms in varying workloads. In *Proceedings of the 2017 26th international conference on computer communication and networks (icccn)* (pp. 1–6). IEEE.

Quan, B. L. Y., Wahab, N. H. A., Fadila, J. N., Aun, Y., Luk, S. K. L., & Wong, K. Y. (2024). The Frontier of Blockchain Privacy: Development of a Private Ethereum Network. In *Proceedings of the 2024 ieee 14th symposium on computer applications & industrial electronics (iscaie)* (pp. 117–122). IEEE.

Rizal, S., & Kim, D.-S. (2025). Enhancing blockchain consensus mechanisms: A comprehensive survey on machine learning applications and optimizations. *Blockchain: Research and Applications*, *6*(4), 100302.

Saeed, H., Malik, H., Bashir, U., Ahmad, A., Riaz, S., Ilyas, M., Bukhari, W. A., & Khan, M. I. A. (2022). Blockchain technology in healthcare: A systematic review. *PLoS ONE*, *17*, e0266462.

Sajna, S., Pillai, M. J., & Rajan, G. (2025). Enhancing healthcare consensus mechanism—A reputation integrated variant of PBFT (BR-PBFT). *PLOS ONE*. https://doi.org/10.1371/journal.pone.0336039.

Smetanin, S., Ometov, A., Komarov, M., Masek, P., & Koucheryavy, Y. (2020). Blockchain evaluation approaches: State-of-the-art and future perspective. *Sensors*, *20*, 3358.

Sunyaev, A., & Sunyaev, A. (2020). Distributed ledger technology. In *Internet computing: Principles of distributed systems and emerging internet-based technologies* (pp. 265–299). Cham, Switzerland: Springer.

Taherdoost, H. (2023). Smart contracts in blockchain technology: A critical review. *Information*, *14*, 117.

Thakkar, P., Nathan, S., & Viswanathan, B. (2018). Performance benchmarking and optimizing hyperledger fabric blockchain platform. In *Proceedings of the 2018 ieee 26th international symposium on modeling, analysis, and simulation of computer and telecommunication systems (mascots)* (pp. 264–276). IEEE.

Ucbas, Y., Eleyan, A., Hammoudeh, M., & Alohaly, M. (2023). Performance and scalability analysis of Ethereum and Hyperledger Fabric. *IEEE Access*, *11*, 67156–67167.

Vora, S., Thakkar, N., & Gor, R. (2023). Study of Performance Measures and Throughput of Raft Consensus Algorithm. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, *11*, 862–869.

Vukolić, M. (2016). The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *International workshop on open problems in network security* (pp. 112–125). Springer.

Wang, J. (2026). *Evmbench: Evaluating ai agents on smart contract security*. arXiv:2603.04915.

Wu, H., Yao, Q., Liu, Z., Huang, B., Zhuang, Y., Tang, H., & Liu, E. (2024). Blockchain for finance: A survey. *IET Blockchain*, *10*, 101–123.

Wüst, K., & Gervais, A. (2018). Do you need a blockchain? In *Proceedings of the 2018 crypto valley conference on blockchain technology (cvcbt)* (pp. 45–54). IEEE.

Yasaweerasinghelage, R., Staples, M., & Weber, I. (2017). Predicting latency of blockchain-based systems using architectural modelling and simulation. In *Proceedings of the 2017 ieee international conference on software architecture (icsa)* (pp. 253–256). IEEE.

Zhang, W., & Anand, T. (2022). Ethereum architecture and overview. In *Blockchain and ethereum smart contract solution development: Dapp programming with solidity* (pp. 209–244). Berlin/Heidelberg, Germany: Springer.

Zheng, K., Liu, Y., Dai, C., Duan, Y., & Huang, X. (2018). Model checking PBFT consensus mechanism in healthcare blockchain network. In *Proceedings of the 2018 9th international conference on information technology in medicine and education (itme)* (pp. 877–881). IEEE.

Zheng, K., Yao, X., Zhang, Z., & Fang, L. (2022). Model Checking the Reliability of Blockchain-based Edge Computing Network. In *Proceedings of the 2022 21st international symposium on distributed computing and applications for business engineering and science (dcabes)* (pp. 246–249). IEEE.

Zheng, Z., Xie, S., Dai, H.-N., Chen, X., & Wang, H. (2018). Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, *14*, 352–375.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.