



A Single-Node Performance Evaluation of Hash-Chain and Dependency-Graph Scheduling for Simulated Smart-Contract Workloads in Organic Certification

Yuni Dwi Anggraeni✉

(Universitas Budi Luhur, Jakarta, Indonesia)

Arum Prasetyajati

(Universitas Budi Luhur, Jakarta, Indonesia)

Akhmad Unggul Priantoro

(Universitas Budi Luhur, Jakarta, Indonesia)

OPEN ACCESS

ARTICLE HISTORY

Received: June 7, 2026

Revised: July 16, 2026

Accepted: August 20, 2026

KEYWORDS

Directed acyclic graph;
Parallel execution;
Performance
benchmarking;
Smart contracts;
Transaction scheduling.

ABSTRACT

Purpose – This study evaluates when dependency-aware parallel scheduling improves a single-node simulated smart-contract workload for organic certification while separating scheduling effects from hash-chain and dependency-graph record representation.

Methods – A deterministic Python rule engine evaluates 18 certification tasks per submission. The same transactions were executed under four conditions: hash-chain sequential (HS), hash-chain dependency-aware parallel (HP), dependency-graph sequential (DS), and dependency-graph dependency-aware parallel (DP). Main experiments varied 18–900 transactions and 1–3,000 iterative SHA-256 rounds, with five warm-ups and 30 paired technical repetitions. Worker, graph-shape, conflict, functional, integrity, and focused reconciliation tests were also performed.

Findings – Representation-only ratios remained close to unity, whereas scheduling produced the dominant performance effect. At 3,000 rounds, HP/HS and DP/DS speedups were 1.680× and 1.620× at 450 transactions and 1.564× and 1.554× at 900 transactions. The focused reconciliation run showed that eight workers minimized makespan for both 180 and 900 transactions, although per-worker efficiency declined as worker count increased. All 12 functional and integrity tests passed.

Research Implications – Dependency information is useful primarily as a scheduling mechanism; the observed speedup should not be interpreted as evidence that a distributed DAG ledger is intrinsically faster than a hash chain.

Originality – The study provides a carefully documented 2 × 2 single-node benchmark that isolates record representation from execution policy for a simulated certification smart-contract workload.

Correspondence Author: ✉ 2211602079@student.budiluhur.ac.id

To cite this article : Anggraeni, Y. D., Prasetyajati, A., & Unggul, A. (2026). A Single-Node Performance Evaluation of Hash-Chain and Dependency-Graph Scheduling for Simulated Smart-Contract Workloads in Organic Certification. *Journal of Deep Learning, Computer Vision and Digital Image Processing*, 4(3), 107–121. <https://doi.org/10.61255/decoding.v4i3.1777>

This is an open access article under the CC BY-SA license



INTRODUCTION

Organic agriculture is increasingly associated with environmentally responsible production, food safety, and long-term resource management [1]. Consumer willingness to pay a premium for organic food is influenced by perceived value and attitudes [2], while trust in organic certification depends on the recognition and credibility of certification schemes [3]. In Indonesia, the growth of organic production increases the practical need for certification workflows that can preserve evidence, identify failed requirements, and support later audit [4]. Organic certification may also strengthen market access and price realization when credible certification information can be connected to digital channels [5]. These conditions motivate certification systems that are not only functionally correct but also able to process repeated rule-evaluation tasks efficiently and reproducibly.

A certification workflow is not a single yes-or-no operation. It combines applicant identity, answers to compliance questions, supporting documents, prerequisite relationships, rule evaluation, integrity checks, and a final certification decision. A digital implementation must therefore distinguish the application logic that evaluates requirements from the data structure used to preserve audit records. This distinction is important for the present study because performance claims can otherwise be attributed to a ledger label even when the actual cause is the execution policy used by the local program.

For an organic-certification application, traceability also has a different operational meaning from generic supply-chain traceability. The system must preserve which requirement was evaluated, which evidence item supported it, which prerequisite had already been satisfied, and whether a failed requirement prevented the final application from being approved. This creates a workload with both audit requirements and explicit task dependencies. Cryptographic record linkage is useful for later integrity verification, but it does not by itself determine whether independent validation tasks should execute serially or concurrently. Treating those concerns separately makes it possible to ask a narrower and more defensible performance question.

Blockchain-based agriculture and certification studies have shown that cryptographically linked records can support traceability, auditability, and cross-party information sharing [6], [7], [8], [9]. General blockchain benchmarking also demonstrates that performance depends strongly on implementation details, workload characteristics, and execution design rather than on the word blockchain itself [10], while scalability studies also emphasize implementation-dependent trade-offs in sharding and decentralized storage [11]. Hyperledger Fabric, for example, separates transaction execution, ordering, validation, and commitment within a permissioned architecture [12]. These studies are relevant to certification because they show that record integrity and application execution are related but technically separable concerns.

Directed acyclic graph (DAG) technologies have likewise been investigated for consensus, ranking, sharding, tip selection, and high-throughput transaction ordering [13], [14], [15], [16], [17], [18], [19], [20], [21], [22]. A recent smart-agriculture study directly compared blockchain and DAG traceability architectures in terms of efficiency and latency [23], while blockchain sharding has also been investigated for Internet-of-Things scalability [24]. However, a distributed DAG ledger is not equivalent to an application dependency graph. The dependency graph used in this study describes prerequisite relationships among certification tasks inside one local workload; it does not implement DAG consensus, tip selection, network finality, or peer-to-peer transaction dissemination.

This separation is especially important because DAG terminology is used at multiple technical layers. In distributed-ledger research, a DAG may describe the transaction or block structure used by a consensus protocol. In a scheduler, by contrast, a DAG may simply encode precedence constraints among tasks that are already ordered or otherwise controlled by the application. The present experiment uses the latter meaning. Consequently, graph depth, width, and read/write conflicts are treated as scheduler inputs, while hash-linked and multi-parent records are treated as alternative local audit representations. The experimental design can then test whether the scheduler benefits from dependency information without implying that a distributed DAG consensus system was implemented.

A second body of literature concerns parallel transaction and smart-contract execution. Ordered blockchain systems are not inherently restricted to serial application execution. Parallel execution can be obtained when independent or non-conflicting transactions are identified through dependency analysis, optimistic concurrency, conflict graphs, or software transactional memory [25], [26]. Block-STM retains a preset transaction order while speculatively exploiting multicore execution and detecting dependencies dynamically [27]. A DAG-based scheduler for Hyperledger Sawtooth also represents transaction dependencies explicitly so that independent smart-contract transactions can execute concurrently [28]. Formal transaction-parallelism work similarly shows that concurrency follows from transaction independence rather than from ledger topology alone [29]. This literature directly motivates separating record representation from execution policy in a controlled benchmark.

The earlier form of this study compared a sequential hash-linked path with a parallel dependency-graph path. Such a comparison changes two factors simultaneously: representation and scheduling policy. Any observed difference could therefore arise from sequential versus parallel execution rather than from hash-chain versus dependency-graph structure. The problem is methodological rather than merely terminological. A fair comparison requires the same workload, rule logic, cryptographic work, transaction order, hardware, timing boundary, and repetition design across all conditions, while representation and execution policy are varied independently.

A related gap concerns the term smart contract. The revised prototype does not deploy Solidity/EVM bytecode, Fabric chaincode, distributed consensus, validator replication, or a production ledger world-state database. Instead, a deterministic Python rule engine models contract-like certification logic at the application level. The ledger or state concept remains the architectural substrate in the literature, but a newly submitted transaction is evaluated before its validated result is committed. The local benchmark therefore models an execute-then-audit-commit lifecycle and is described as a simulated smart-contract workload rather than a deployed smart-contract platform [12]. This framing preserves the application context without extending the empirical claim beyond what the code actually implements.

The organic-certification workload provides a useful controlled case because each submission contains 18 tasks: 13 question nodes and five supporting-document nodes with explicit prerequisites. Every task executes deterministic certification-rule logic, payload hashing, controlled iterative SHA-256 work, and HMAC-SHA256 tag generation. By varying the amount of SHA-256 work, transaction count, worker count, graph shape, and shared-state conflicts, the experiment can identify when parallel coordination costs are amortized by useful computation and whether record topology contributes materially once scheduling is held constant.

The study therefore evaluates four questions. First, does dependency-aware parallel execution outperform sequential execution when record representation is held constant? Second, does hash-chain versus dependency-graph representation materially change performance when execution policy is held constant? Third, at what tested computational intensities does the parallel advantage become statistically consistent for different transaction counts? Fourth, how sensitive are the results to worker count, graph width and depth, and synthetic read/write conflicts? The contribution is an application-specific single-node scheduler benchmark comprising a 2×2 factorial comparison of hash-chain sequential (HS), hash-chain dependency-aware parallel (HP), dependency-graph sequential (DS), and dependency-graph dependency-aware parallel (DP), together with deterministic equivalence checks, functional and integrity tests, sensitivity analyses, and retained run-level measurements.

METHOD

Research Design and Experimental Environment

The study used a controlled quantitative benchmark on one Windows laptop; experimental units were preconstructed certification transactions. Two factors were varied independently: record representation (hash chain or dependency graph) and execution policy (sequential or dependency-aware parallel), producing HS, HP, DS, and DP. Each paired scenario used identical transaction

payloads, rule outcomes, canonical sequence, cryptographic workload, and configuration. Figure 1 summarizes the design and Table 1 the environment.

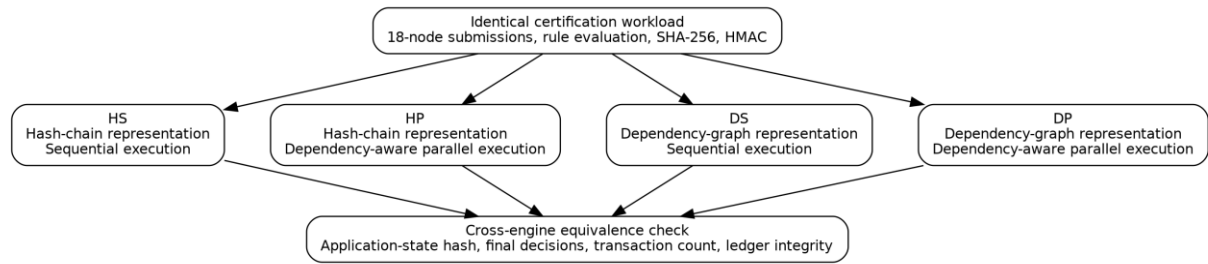


Figure 1. Factorial separation of record representation and execution policy.

Benchmarks ran from a local non-synchronized directory under a fixed AC-powered performance state. Interactive workloads and Streamlit were excluded from timed runs. All conditions used the same Python environment and captured package versions.

Table 1. Experimental hardware and software environment.

Parameter	Specification
Processor	AMD Ryzen 5 6600H (3.30 GHz); 12 logical CPUs
Memory	16 GB RAM
Operating system	Windows 11, 64-bit (build reported as 10.0.26200)
Python	3.13.3, 64-bit AMD64
Key packages	Streamlit 1.61.1; pandas 2.3.3; NumPy 2.5.1; SciPy 1.18.0; Matplotlib 3.11.1; graphviz 0.21; psutil 7.2.2
Parallel executor	concurrent.futures.ProcessPoolExecutor with Windows spawn context

Instrumentation, Transaction Model, and Dependency Structure

Each transaction contained identifiers, canonical sequence, payload, parents, read/write sets, and deterministic rule inputs. The rule engine produced PASS or FAIL. Serialization used canonical JSON with lexicographically sorted keys, compact comma-and-colon separators, UTF-8 encoding, and non-ASCII preservation. The payload was hashed once with SHA-256 to a 32-byte digest represented as 64 hexadecimal characters. After rule evaluation, the iterative workload object contained transaction ID, submission ID, question code, payload digest, and status; its measured canonical length was 178–182 bytes. Round 1 hashed this full byte sequence and each subsequent round hashed the preceding 32-byte raw digest. Thus, 3,000 configured rounds comprised one hash of the input plus 2,999 hashes of successive digests; the payload-digest hash was additional. HMAC-SHA256 then used the 64-character ASCII hexadecimal workload digest and one fixed 41-byte UTF-8 benchmark-only key shared across all conditions; no production key management was implemented. Records were committed to either a previous-hash chain or a multi-parent dependency graph. Smart-contract behavior was simulated in Python; no Solidity/EVM, Fabric chaincode, consensus, or replicated contract storage was executed.

The 18-node organic-certification logic is illustrative benchmark logic, not a clause-by-clause implementation of an authoritative certification standard or a legally valid certification procedure. The domain supplies prerequisite relations, supporting-document dependencies, deterministic PASS/FAIL processing, and an auditable final decision. Functional tests therefore establish correctness relative to the programmed rules, not regulatory or domain validity. The balanced template contained 18 nodes and 22 prerequisite edges with a nine-node critical path; Figure 2 shows its full parent structure. Deep and wide templates were used only for graph-sensitivity analysis (Table 2).

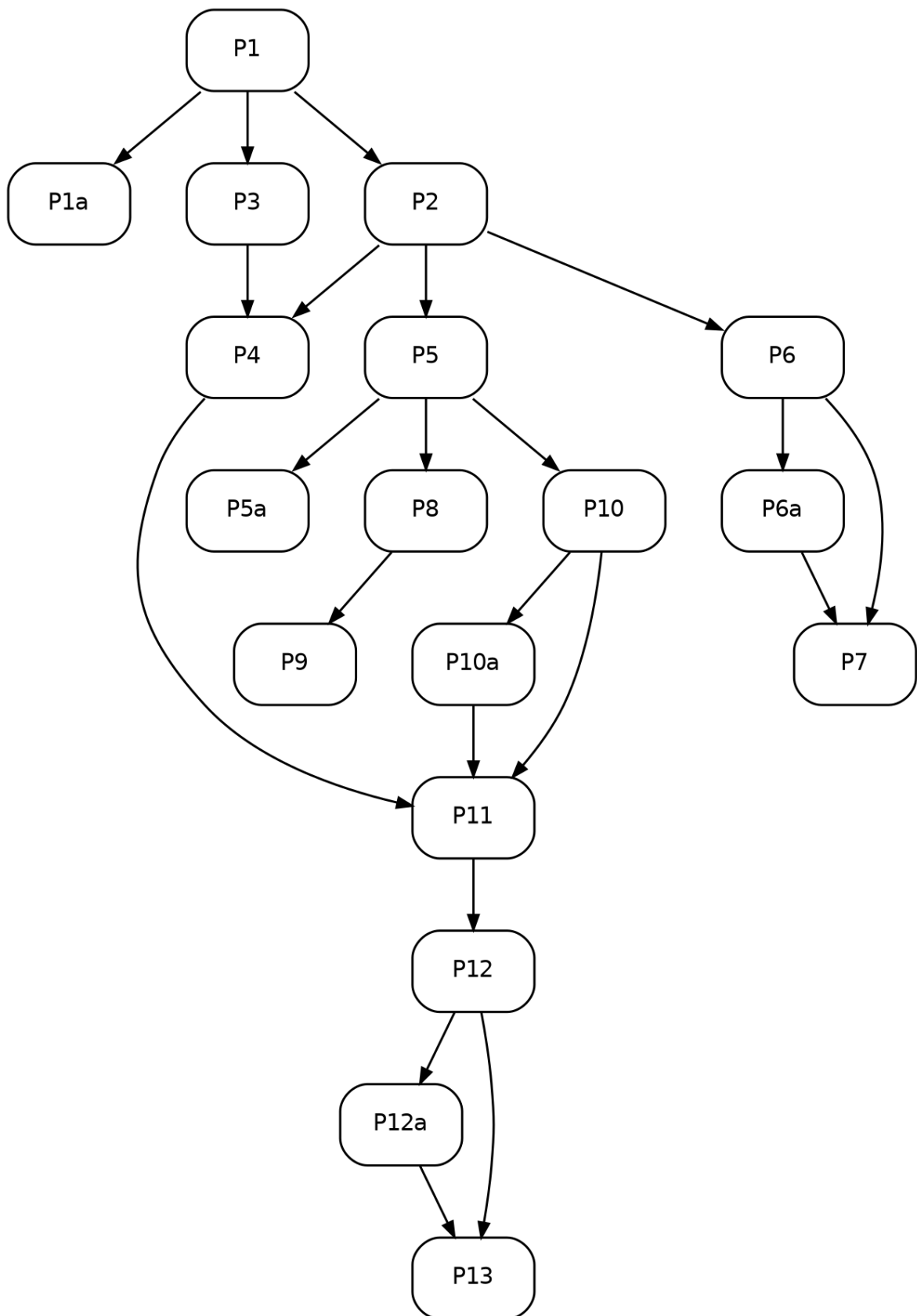


Figure 2. Balanced 18-node application dependency graph used in the main benchmark.

Table 2. Structural metrics of the three dependency templates.

Graph	Nodes	Edges	Max depth	Max width	Critical path (nodes)	Interpretation
Deep	18	17	17	1	18	Near-linear dependency chain
Balanced	18	22	8	4	9	Certification prerequisite template
Wide	18	17	1	17	2	P1 followed by maximally independent children

Execution Conditions and Deterministic Equivalence

Sequential execution followed canonical order. Dependency-aware parallel execution repeatedly formed a ready set whose parents were satisfied, selected up to the worker limit, and serialized read/write conflicts. Ready tasks were submitted to ProcessPoolExecutor and completed as a wave before the next ready set. This reflects transaction-independence approaches to parallel execution [25], [26]–[29].

Table 3. Four-condition factorial experimental design.

Code	Representation	Execution policy	Workers	Commit/ordering
HS	Hash chain	Sequential	1	Canonical sequential commit
HP	Hash chain	Dependency-aware parallel	p	Parallel evaluation; canonical sequential hash-chain commit
DS	Dependency graph	Sequential	1	Canonical evaluation; multi-parent graph commit
DP	Dependency graph	Dependency-aware parallel	p	Parallel evaluation; multi-parent graph commit

Cross-engine equivalence required the same application-state hash, final certification decisions, and transaction count, while each representation passed its own integrity check. Ledger roots were not expected to match because hash-chain and dependency-graph records encode different links.

Procedures and Timing Boundary

ProcessPoolExecutor instances were created and primed before measurement. Scenario order within each paired block was randomized with a fixed seed, and each condition used five warm-ups followed by 30 technical repetitions. A focused reconciliation repeated the disputed 180- and 900-transaction conditions at 3,000 SHA-256 rounds with 1, 2, 4, and 8 workers. Within each block, the same preconstructed workload and primed pools were used; HS and DS ran once and HP/DP at each worker count. Consequently, each four-worker HP/DP observation was generated once and used consistently in both the reconciled main comparison and worker-scaling analysis.

Table 4. Final benchmark protocol.

Study	Transactions	SHA-256 rounds	Workers	Graph	Conflict setting	Measured repetitions
Main benchmark	18, 180, 450, 900	1, 100, 500, 650, 750, 850, 1,000, 1,500, 3,000	4	Balanced	0%	30
Worker scaling	180, 900	100, 1,000, 3,000	1, 2, 4, 8	Balanced	0%	30
Graph sensitivity	180, 900	100, 1,000, 3,000	4	Deep, balanced, wide	0%	30
Conflict sensitivity	180, 900	100, 1,000, 3,000	4	Balanced	0%, 25%, 50% configured	30
Focused reconciliation	180, 900	3,000	1, 2, 4, 8	Balanced	0%	30

Timing started immediately before sequential or dependency-aware scheduling and ended after representation-specific commit and integrity verification. It included scheduling/submission, rule evaluation, SHA-256/HMAC, result collection, record construction, commit, and integrity checks, but excluded workload/graph construction, pool startup, Streamlit, CSV writing, application-state hashing, and final-decision aggregation.

Analysis Plan and Performance Metrics

The primary outcome was batch makespan, calculated with the high-resolution timer in (1). Because paired engines processed the same transaction count N , (2) gives a secondary local processing rate; it is not interpreted as confirmed distributed-ledger throughput.

$$M_{ms} = (t_{end} - t_{start}) / 1,000,000 \quad (1)$$

The secondary local processing rate was:

$$R = N / (M_{ms} / 1,000) \quad (2)$$

Paired speedup was T_1/T_p in (3) and parallel efficiency S_p/p in (4). Each condition reports mean, median, sample standard deviation, minimum, maximum, P95, and a 95% bootstrap confidence interval for the mean using 5,000 fixed-seed resamples. Paired speedups were aggregated within pair identifiers. Conservative crossover was the first sampled SHA-256 intensity whose 95% speedup interval lay entirely above 1.

$$S_p = T_1 / T_p \quad (3)$$

and parallel efficiency is:

$$E_p = S_p / p \quad (4)$$

Functional and Integrity Tests

Table 5 covers certification outcomes, tampering, missing parents, cycles, HMAC verification, duplicates/replay, conflict serialization, graph-record tampering, and deterministic replay. These tests demonstrate implementation correctness relative to the programmed benchmark rules, not regulatory validity. The benchmark is limited to local CPU execution and audit-record construction; it excludes network communication, distributed ordering, consensus, replication, validator behavior, gas accounting, and production smart-contract runtimes. Source code and tests are supplied as supplementary material; configurations, environment metadata, analysis scripts, and run-level CSVs were retained as the experiment audit record.

Table 5. Functional and integrity test matrix.

ID	Case	Expected	Observed	Status
FT01	All required nodes valid	APPROVED	APPROVED	PASS
FT02	Required nodes include failures	PENDING	PENDING	PASS
FT03	Payload digest modified after commit	Tampering detected	Detected	PASS
FT04	Previous hash modified	Hash chain invalid	Invalid	PASS
FT05	Dependency parent missing	Rejected	Rejected	PASS
FT06	Dependency cycle introduced	Cycle detected	Detected	PASS
FT07	HMAC verified with wrong key	Authentication failure	Failure	PASS
FT08	Duplicate transaction identifier	Duplicate detected	Detected	PASS
FT09	Committed transaction replayed	Replay detected	Detected	PASS
FT10	Concurrent write conflicts	Conflicts serialized	Serialized	PASS
FT11	Committed graph record modified	Modification detected	Detected	PASS
FT12	Identical input replayed	Identical state and ledger roots	Identical	PASS

RESULTS AND DISCUSSION

Correctness and Experimental Validity

All 12 functional and integrity tests in Table 5 passed. The four original performance studies produced 11,520 measured engine runs: 4,320 in the main benchmark, 2,880 in worker scaling, 2,160 in graph sensitivity, and 2,160 in conflict sensitivity. The focused reconciliation added 600 measured runs, giving 12,120 measured runs across the complete revised evidence base. Every evaluated condition passed the relevant application-state equivalence and representation-specific ledger-integrity checks. Thus, performance comparisons were made only among executions that produced the same canonical application state, final decisions, and transaction count. Representation-specific ledger roots differed by design because their hash envelopes are structurally different.

Main Factorial Benchmark and Crossover Behavior

Table 6. Main benchmark at 3,000 SHA-256 rounds and conservative crossover thresholds.

Tx	HS ms	HP ms	DS ms	DP ms	HP/HS speedup [95% CI]	DP/DS speedup [95% CI]	HP first CI > 1	DP first CI > 1
18	48.18	43.41	48.75	43.60	1.114 [1.091, 1.138]	1.125 [1.093, 1.157]	3,000	3,000
180	479.86	296.63	478.78	296.50	1.639 [1.571, 1.719]	1.642 [1.564, 1.737]	850	750
450	1,241.30	741.60	1,220.36	754.45	1.680 [1.636, 1.731]	1.620 [1.594, 1.650]	850	750
900	2,412.95	1,543.43	2,406.24	1,549.99	1.564 [1.547, 1.583]	1.554 [1.539, 1.570]	850	850

Note. Speedup values are means of within-repetition paired ratios (T_{seq}/T_{par}), not ratios of the aggregated mean makespans; therefore, displayed speedups may differ slightly from direct division of the reported means.

Figure 3 shows the paired speedup curves across computational intensity. At one and 100 SHA-256 rounds, both parallel conditions were substantially slower because task-submission, inter-process communication, wave synchronization, and result-collection costs dominated useful work. The crossover shifted with batch size. For 18 transactions, neither HP nor DP achieved a confidence interval wholly above 1 until 3,000 rounds. For 180 and 450 transactions, DP reached the conservative criterion at 750 rounds and HP at 850 rounds. For 900 transactions, both reached it at 850 rounds. The 3,000-round points for 180 and 900 transactions use the focused reconciliation measurements so that the four-worker values are identical wherever the same condition appears in Tables 6 and 7; the crossover thresholds remain those observed in the original computational-intensity sweep.

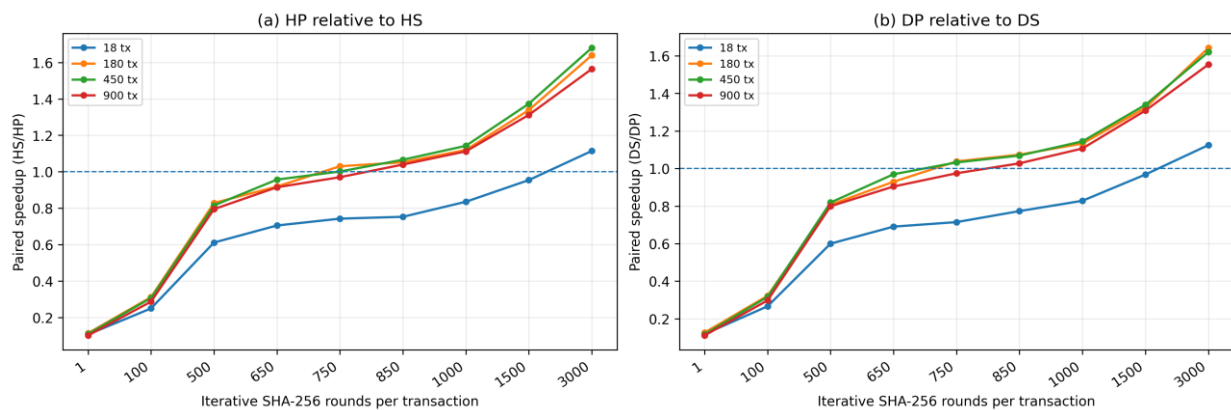


Figure 3. Paired speedup across computational intensity: (a) HP relative to HS and (b) DP relative to DS. The dashed line marks speedup = 1.

At 3,000 rounds, HP/HS speedup was 1.114× at 18 transactions, 1.639× at 180, 1.680× at 450, and 1.564× at 900. DP/DS speedup was 1.125×, 1.642×, 1.620×, and 1.554× for the same batch sizes. The result therefore remains strongly favorable to dependency-aware parallel execution for the larger compute-intensive batches, but it is not monotonic with transaction count. The reduction at 900 transactions indicates that process coordination, memory behavior, synchronization, and available CPU resources impose a practical ceiling on end-to-end scaling in this hardware environment.

The paired design is important for interpreting these values. Each HP observation is compared with the HS observation generated from the same scenario and pair identifier, and the same principle is used for DP relative to DS. Consequently, the reported confidence intervals characterize repeated technical variation within the controlled machine rather than variation among independent hardware populations. The crossover threshold is therefore an empirical property of this implementation and processor configuration. It identifies where measured useful work first exceeded coordination overhead with conservative uncertainty, but it should not be treated as a universal SHA-256 threshold for other systems.

Representation Effect Versus Execution-Policy Effect

The factorial design changes the interpretation relative to a direct 'blockchain versus DAG' comparison. When execution policy was held constant, representation-only ratios remained close to unity. At 3,000 rounds, the sequential representation ratio HS/DS was approximately 0.988, 1.002, 1.017, and 1.003 for 18, 180, 450, and 900 transactions, respectively. The corresponding parallel representation ratios likewise remained close to 1. Across the full main benchmark, some small representation differences had confidence intervals that excluded 1, but the direction was not consistent and the magnitude was usually only a few percent. In contrast, sequential-versus-parallel ratios changed from large disadvantages at fine task granularity to approximately 1.55–1.68× advantages at high granularity. The dominant experimental factor is therefore execution policy interacting with task granularity, not record topology alone.

This result is consistent with the broader parallel smart-contract literature, while the present implementation remains a simulated single-node rule-engine workload rather than a deployed smart-contract platform. Ordered blockchain execution can also exploit multicore hardware when non-conflicting transactions are identified, whether through explicit dependency graphs, optimistic concurrency, or STM-based speculation [25],[26]. The present dependency graph is simpler than Block-STM and does not perform speculative abort/re-execution; nevertheless, the comparison reinforces the same methodological lesson: parallelism should be attributed to the scheduler and conflict model rather than to the mere presence of a DAG-shaped record structure.

The factorial result also explains why a direct label-level comparison can be misleading. In the revised experiment, HP and DP receive nearly the same benefit when they use the same dependency-aware scheduler, even though their audit records are linked differently. Conversely, HS and DS remain similar when both execute sequentially. This pattern does not imply that ledger structure is irrelevant in production distributed systems, where consensus, network propagation, storage layout, conflict validation, and finality can dominate end-to-end behavior. It shows only that, within this single-node benchmark, the local scheduler and the computational granularity of each task account for the large observed change in makespan.

Worker Scaling

Worker-scaling speedup was calculated within repetition pairs as T_1/T_p and averaged before parallel efficiency (Sp/p) was computed; therefore, efficiency may differ slightly from values obtained by dividing the displayed mean makespans.

Table 7. Reconciled worker scaling at 3,000 SHA-256 rounds.

Tx	Engine	1 worker ms	2 workers ms	4 workers ms	8 workers ms	Lowest makespan at	Efficiency (4w; 8w)
180	HP	683.00	409.33	296.63	209.93	8	58.4%; 41.2%
180	DP	684.18	405.27	296.50	219.31	8	58.7%; 39.5%

Tx	Engine	1 worker ms	2 workers ms	4 workers ms	8 workers ms	Lowest makespan at	Efficiency (4w; 8w)
900	HP	3,582.37	2,174.06	1,543.43	940.04	8	58.1%; 47.7%
900	DP	3,593.54	2,140.60	1,549.99	934.51	8	58.0%; 48.1%

The focused reconciliation produced a consistent worker-scaling series for the overlapping 3,000-round conditions. For 180 transactions, HP makespan decreased from 683.00 ms with one worker to 296.63 ms with four workers and 209.93 ms with eight workers; the 95% bootstrap confidence intervals for the four- and eight-worker means were [284.47, 306.88] and [201.69, 218.27] ms. DP showed the same pattern, decreasing from 684.18 to 296.50 and 219.31 ms, with corresponding four- and eight-worker intervals of [283.53, 307.84] and [210.83, 227.27] ms. For 900 transactions, eight workers also produced the lowest makespan: 940.04 ms for HP [932.96, 946.78] and 934.51 ms for DP [925.00, 943.35]. However, per-worker efficiency declined with additional workers. Four-worker efficiency was approximately 58.4–58.7% for the 180-transaction batch and 58.0–58.1% for 900 transactions, whereas eight-worker efficiency was approximately 39.5–41.2% and 47.7–48.1%, respectively. Thus, additional workers reduced absolute makespan but exhibited diminishing returns. Four workers remain the factorial baseline for the main benchmark to preserve the controlled 2×2 comparison, not because four workers are universally optimal.

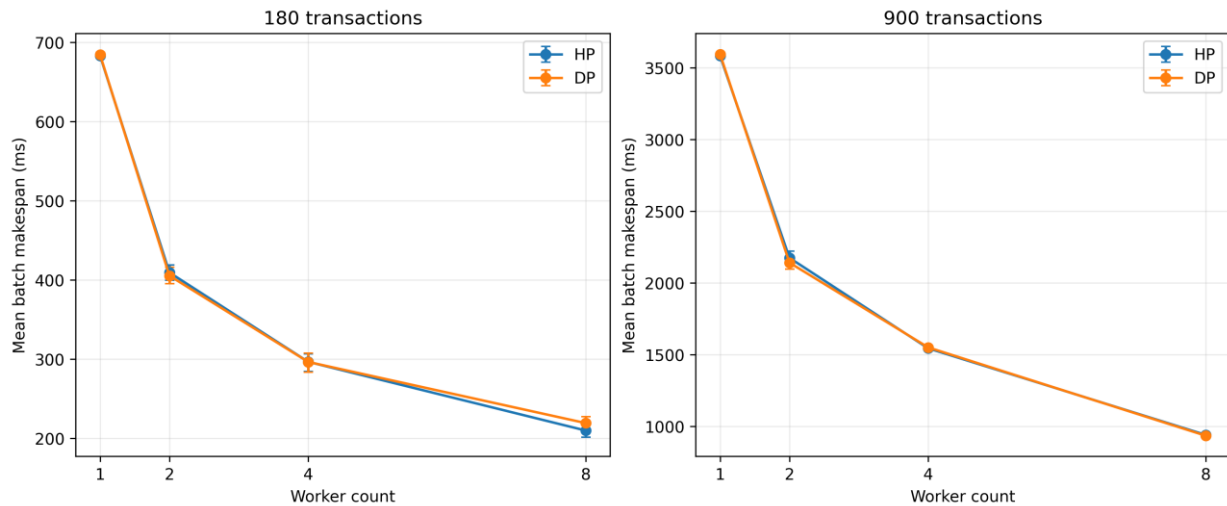


Figure 4. Reconciled worker scaling at 3,000 SHA-256 rounds for (a) 180 and (b) 900 transactions; error bars show 95% bootstrap confidence intervals.

Graph-Structure and Conflict Sensitivity

Graph shape affected the number of scheduler waves and had its clearest effect in the smaller batch. For 180 transactions, the deep, balanced, and wide templates required approximately 54, 48, and 46 waves, respectively. At 3,000 rounds, HP mean makespan was 313.46 ms [308.54, 316.88] for the deep graph, 302.52 ms [296.17, 306.23] for the balanced graph, and 300.93 ms [299.46, 302.64] for the wide graph. DP showed the same general pattern. For 900 transactions, however, the 95% confidence intervals for balanced and wide configurations overlapped substantially for both engines, and their mean differences were small. The sensitivity results therefore support a cautious conclusion: deep dependency structures can increase scheduling cost, but the balanced-versus-wide advantage is small and statistically uncertain once independent submissions already provide abundant cross-submission parallelism.

Synthetic conflict injection increased the number of ready tasks that had to be deferred, but it did not produce a monotonic makespan penalty in the tested patterns. At 180 transactions, serialized conflict events increased from 0 at the 0% setting to 14 at 25% and 56 at 50%; at 900 transactions they increased from 0 to 49 and 272. At 3,000 rounds, 900-transaction HP makespan was 1,512.05 ms [1,496.25, 1,524.42] at 0%, 1,530.02 ms [1,527.33, 1,532.77] at 25%, and 1,516.23 ms [1,499.57, 1,527.84] at 50%. The configured percentages are probabilities used to inject a shared certification-

state resource and are not identical to the observed fraction of serialized transactions. Because the mean response was small and non-monotonic across conflict levels and engines, the experiment does not support a general claim that increasing the configured conflict probability produces a proportional slowdown in this synthetic model.

The observed speedups can also be related quantitatively to the available parallelism. The balanced per-submission template has 18 tasks and a nine-task critical path; independent submissions increase the batch-level ready width beyond four workers. With four workers, the work-bound lower limit is $\text{ceil}(180/4) = 45$ waves for the 180-transaction batch and $\text{ceil}(900/4) = 225$ waves for the 900-transaction batch, compared with observed conflict-free counts of approximately 48 and 228 waves. The scheduler therefore operated only about 6.7% and 1.3% above these lower bounds. Nevertheless, the end-to-end HP/HS and DP/DS policy speedups at 3,000 rounds were only about 1.55–1.64 $\times$ because the timed region also includes process scheduling, inter-process communication, wave barriers, result collection, commit, and integrity verification. This ratio is distinct from worker-scaling speedup within HP or DP: the one-worker parallel engine itself carries dependency-scheduler and process-management overhead that is absent from HS or DS.

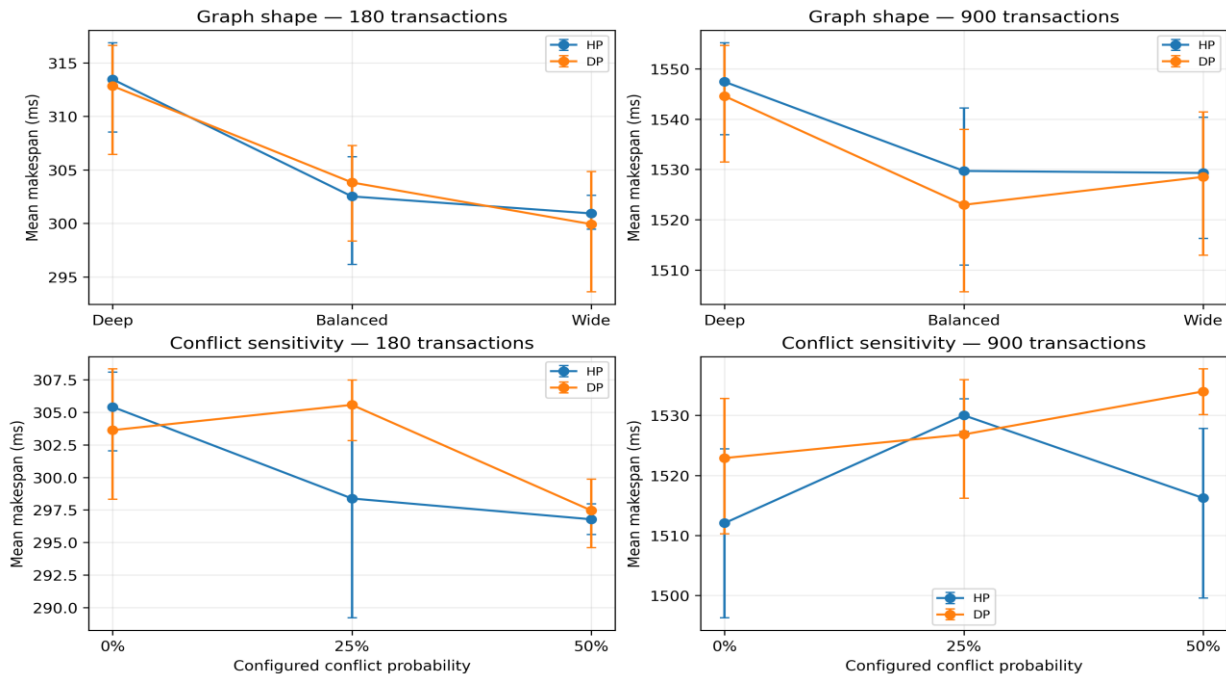


Figure 5. Sensitivity at 3,000 SHA-256 rounds: graph shape for (a) 180 and (b) 900 transactions and configured conflict probability for (c) 180 and (d) 900 transactions; error bars show 95% bootstrap confidence intervals.

For certification-system engineering, explicit dependencies provide two distinct benefits. First, they describe prerequisite relationships among evidence items and rule checks. Second, they give a scheduler enough information to execute independent work concurrently when the per-task cost justifies coordination. These benefits do not require claiming that a dependency graph is itself a faster ledger. The hash-chain representation obtained essentially the same parallel-execution benefit when it used the same dependency-aware scheduler.

The study also supports a conservative interpretation of performance. One-pass or otherwise fine-grained tasks should not be parallelized automatically because process scheduling can cost substantially more than the useful work. Conversely, compute-intensive batches with hundreds of transactions benefit from multicore execution. The reconciliation run showed that eight workers minimized absolute makespan for both tested batch sizes, while parallel efficiency decreased as workers were added. For system designers, the practical implication is to treat dependency-aware scheduling as an adaptive execution mechanism whose benefit depends on task granularity, batch size, worker availability, nonparallel overhead, and the amount of independent work rather than as a fixed property of the ledger representation.

The principal contribution is methodological and auditability-oriented. The study converts an earlier two-path comparison into a 2×2 factorial benchmark that separates record representation from execution policy, adds deterministic cross-engine equivalence, and reports the crossover at which dependency-aware parallelism becomes beneficial under controlled computational intensities. A focused reconciliation resolves the previously inconsistent overlapping four-worker measurements, while worker-scaling, graph-shape, and conflict-sensitivity analyses quantify uncertainty and show how the result changes when available parallelism or contention changes. The work also provides an illustrative domain-specific certification task graph in which failed requirements remain auditable while the final application decision is deterministic. These elements create a clearer baseline for future experiments that may later substitute a real contract runtime or distributed ledger without changing the central comparison logic.

Several limitations remain. The workload is synthetic and CPU-oriented; real certification systems would add database access, document parsing, asymmetric-signature verification, network services, human inspection, and policy-specific validation. The 18-node certification rules and dependency structure are illustrative benchmark abstractions and were not formally validated against an authoritative organic-certification standard or certification-body operating procedure; accordingly, the correctness results concern software behavior relative to the implemented rules rather than regulatory certification validity. The simulated smart-contract logic is a deterministic Python rule engine rather than deployed Solidity/EVM bytecode, chaincode, or another replicated contract runtime. Although the prototype uses local hash-chain and dependency-graph audit representations, it does not maintain a production ledger world-state database or execute rules by reading and updating deployed contract storage. The conflict model injects a shared read/write resource probabilistically rather than reproducing all forms of smart-contract contention. Finally, the benchmark uses one Windows laptop, no explicit CPU affinity, and no distributed network. Accordingly, the results cannot be generalized to production blockchain or DAG consensus platforms, confirmed transaction throughput, or end-to-end contract execution across validators.

Future work should repeat the factorial design on additional processors and operating systems, introduce realistic database and document I/O, incorporate asymmetric cryptography, validate the certification logic against an authoritative domain procedure, and derive read/write conflicts from observed certification workflows rather than probabilistic injection. A second stage should replace the simulated rule engine with an established smart-contract runtime and compare dependency-aware scheduling with speculative engines such as Block-STM under equivalent transaction semantics. Multi-node experiments would then be required before making claims about distributed consensus, network finality, or confirmed transaction throughput. The present study is supported by supplementary source code and tests together with retained fixed configuration values, environment metadata, analysis scripts, and run-level measurements.

CONCLUSION

This study re-evaluated a single-node simulated smart-contract workload for organic-certification transaction processing with a factorial design that separates record representation from execution policy. The same 18-node illustrative certification workload was executed as a hash chain or dependency graph and under sequential or dependency-aware parallel scheduling. The main benchmark covered 18–900 transactions, nine SHA-256 intensities, five warm-up runs, and 30 paired technical repetitions per scenario, while additional experiments varied worker count, graph structure, and synthetic read/write conflicts. A focused reconciliation repeated the overlapping 180- and 900-transaction, 3,000-round conditions so that the same four-worker measurements are used consistently across the main-comparison and worker-scaling results. All evaluated conditions produced equivalent application states and valid representation-specific audit structures.

The results do not support a general claim that a DAG representation is intrinsically faster than a linear hash-chain representation. With the same scheduler, representation-only performance ratios remained near unity. The substantial effect came from execution policy: dependency-aware parallel scheduling was slower for fine-grained tasks but became consistently advantageous once useful

computation and batch size were sufficient to amortize process coordination. At 3,000 rounds, HP/HS and DP/DS speedups were 1.680× and 1.620× at 450 transactions and 1.564× and 1.554× at 900 transactions. The conservative crossover occurred at 750–850 rounds for most 180–900 transaction scenarios and at 3,000 rounds for the 18-transaction batch.

Worker scaling, graph sensitivity, and conflict injection further show that exploitable parallelism depends on resource count and workload structure. Eight workers produced the lowest reconciled makespan for both 180 and 900 transactions, but efficiency declined as additional workers were used. Deep graphs tended to increase scheduling cost, whereas balanced-versus-wide differences were small and often uncertain for the larger batch. Synthetic conflicts increased serialization without a consistent monotonic makespan penalty. The balanced graph also operated close to its four-worker wave lower bound, indicating that the approximately 1.55–1.64× end-to-end policy speedup was constrained mainly by process, synchronization, communication, commit, and integrity-check overhead rather than by insufficient graph width. The contribution is therefore a carefully documented single-node scheduler benchmark and correctness framework for an illustrative simulated smart-contract rule-processing workload in organic certification, not a deployed smart-contract platform or distributed-ledger consensus comparison.

ACKNOWLEDGMENT

The authors have no acknowledgments to declare.

AUTHOR CONTRIBUTION STATEMENT

YDA Conceptualization, Methodology, Software, Validation, Formal analysis, Writing – original draft. AP Supervision, Validation, Writing – review & editing. AU Supervision, Writing – review & editing. All authors have read and approved the final version of the manuscript.

AI DISCLOSURE STATEMENT

The authors used ChatGPT by OpenAI as a supporting tool during manuscript preparation and revision, primarily for language refinement, technical support. All benchmark experiments were conducted by the authors using the reported hardware and configuration. The authors reviewed and verified the resulting data, analyses, interpretations, and final manuscript, and take full responsibility for the content of this publication.

REFERENCES

- [1] S. Sharma, 'Organic Agriculture for Sustainable Food Systems: A Comprehensive Review of Benefits and Constraints', *Turkish J. Agric. - Food Sci. Technol.*, vol. 12, no. 8, pp. 1476–1481, 2024, doi: 10.24925/turjaf.v12i8.1476-1481.6482.
- [2] T. Hu, A. Al Mamun, M. N. H. Reza, M. Wu, and Q. Yang, 'Examining Consumers' Willingness to Pay Premium Price for Organic Food', *Humanit. Soc. Sci. Commun.*, vol. 11, no. 1, p. 1249, 2024, doi: 10.1057/s41599-024-03789-6.
- [3] B. Murphy, M. Martini, A. Fedi, B. L. Loera, C. T. Elliott, and M. Dean, 'Consumer Trust in Organic Food and Organic Certifications in Four European Countries', *Food Control*, vol. 133, p. 108484, 2022, doi: 10.1016/j.foodcont.2021.108484.
- [4] W. David, S. Alkausar, and B. Widyarti, *Statistik Pertanian Organik Indonesia*. Jakarta, Indonesia: Universitas Bakrie Press, 2023.
- [5] L. Zhang, D. Liu, Q. Yin, and J. Liu, 'Organic Certification, Online Market Access, and Agricultural Product Prices: Evidence from Chinese Apple Farmers', *Agriculture*, vol. 14, no. 5, p. 669, 2024, doi: 10.3390/agriculture14050669.
- [6] S. Tegeltija *et al.*, 'Blockchain Framework for Certification of Organic Agriculture Production', *Sustainability*, vol. 14, no. 19, p. 11823, 2022, doi: 10.3390/su141911823.
- [7] Y. Huang, X. Li, L. Xu, and Y. Ma, 'Digital Traceability in Horticulture: A Systematic Review of

- Edge-Cloud-Blockchain-Terminal (ECBT) Integration with IoT and AI Technologies', *Front. Blockchain*, vol. 8, p. 1636627, 2025, doi: 10.3389/fbloc.2025.1636627.
- [8] T. H. Pranto, A. A. Noman, A. Mahmud, and A. K. M. B. Haque, 'Blockchain and Smart Contract for IoT Enabled Smart Agriculture', *PeerJ Comput. Sci.*, vol. 7, p. e407, 2021, doi: 10.7717/peerj-cs.407.
- [9] S. A. Bhat, N.-F. Huang, I. B. Sofi, and M. Sultan, 'Agriculture-Food Supply Chain Management Based on Blockchain and IoT: A Narrative on Enterprise Blockchain Interoperability', *Agriculture*, vol. 12, no. 1, p. 40, 2022, doi: 10.3390/agriculture12010040.
- [10] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, 'BLOCKBENCH: A Framework for Analyzing Private Blockchains', in *Proceedings of the 2017 ACM International Conference on Management of Data*, in SIGMOD '17. ACM, 2017, pp. 1085–1100. doi: 10.1145/3035918.3064033.
- [11] A. L. Bulgakov, A. V Aleshina, S. D. Smirnov, A. D. Demidov, M. A. Milyutin, and Y. Xin, 'Scalability and Security in Blockchain Networks: Evaluation of Sharding Algorithms and Prospects for Decentralized Data Storage', *Mathematics*, vol. 12, no. 23, p. 3860, 2024, doi: 10.3390/math12233860.
- [12] E. Androulaki *et al.*, 'Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains', in *Proceedings of the Thirteenth EuroSys Conference*, in EuroSys '18. ACM, 2018, pp. 1–15. doi: 10.1145/3190508.3190538.
- [13] B. Cao *et al.*, 'Performance Analysis and Comparison of PoW, PoS and DAG Based Blockchains', *Digit. Commun. Networks*, vol. 6, no. 4, pp. 480–485, 2020, doi: 10.1016/j.dcan.2019.12.001.
- [14] F. Kahmann, F. Honecker, J. Dreyer, M. Fischer, and R. Tönjes, 'Performance Comparison of Directed Acyclic Graph-Based Distributed Ledgers and Blockchain Platforms', *Computers*, vol. 12, no. 12, p. 257, 2023, doi: 10.3390/computers12120257.
- [15] Y. Bai, S. Lee, and S.-H. Seo, 'A Survey on Directed Acyclic Graph-Based Blockchain in Smart Mobility', *Sensors*, vol. 25, no. 4, p. 1108, 2025, doi: 10.3390/s25041108.
- [16] J. Huang, C. Liu, and J. Harding, 'Research on Blockchain Architecture and Operating Principles Based on H-DAG', *Symmetry (Basel)*, vol. 15, no. 7, p. 1361, 2023, doi: 10.3390/sym15071361.
- [17] T. Noreen, Q. Xia, and M. Z. Haider, 'Advanced DAG-Based Ranking (ADR) Protocol for Blockchain Scalability', *Comput. Mater. & Contin.*, vol. 75, no. 2, pp. 2593–2613, 2023, doi: 10.32604/cmc.2023.036139.
- [18] X. Lu and C. Jiang, 'TEEDAG: A High-Throughput Distributed Ledger Based on TEE and Directed Acyclic Graph', *Electronics*, vol. 12, no. 11, p. 2393, 2023, doi: 10.3390/electronics12112393.
- [19] X. Fu, H. Wang, P. Shi, and X. Zhang, 'Teegraph: A Blockchain Consensus Algorithm Based on TEE and DAG for Data Sharing in IoT', *J. Syst. Archit.*, vol. 122, p. 102344, 2022, doi: 10.1016/j.sysarc.2021.102344.
- [20] W. Zheng *et al.*, 'Data Management Method for Building Internet of Things Based on Blockchain Sharding and DAG', *Internet Things Cyber-Physical Syst.*, vol. 4, pp. 217–234, 2024, doi: 10.1016/j.iotcps.2024.01.001.
- [21] R. Xiao, Y. Cao, and B. Xia, 'Adaptive Tip Selection for DAG-Shard-Based Federated Learning with High Concurrency and Fairness', *Sensors*, vol. 25, no. 1, p. 19, 2025, doi: 10.3390/s25010019.
- [22] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, 'Bullshark: DAG BFT Protocols Made Practical', in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, Association for Computing Machinery, 2022, pp. 2705–2718. doi: 10.1145/3548606.3559361.

- [23] A. Villafranca *et al.*, 'Comparing Blockchain and DAG Technologies for Smart Agriculture Traceability in Terms of Efficiency and Latency', *Simul. Model. Pract. Theory*, vol. 142, p. 103131, 2025, doi: 10.1016/j.simpat.2025.103131.
- [24] J. Wang, Y. Li, Y. Wu, W. Zheng, S. Zhou, and X. Xiong, 'Blockchain Sharding Scheme Based on Generative AI and DRL: Applied to Building Internet of Things', *Internet Things Cyber-Physical Syst.*, vol. 4, pp. 333–349, 2024, doi: 10.1016/j.iotcps.2024.11.001.
- [25] T. Xu and Y. Zhang, 'A Parallel Execution Model for Permissioned Blockchain Systems', in *Proceedings of the 2022 5th International Conference on Blockchain Technology and Applications*, Association for Computing Machinery, 2022, pp. 83–87. doi: 10.1145/3581971.3581983.
- [26] P. S. Anjana, S. Kumari, S. Peri, S. Rathor, and A. Somani, 'An Efficient Framework for Optimistic Concurrent Execution of Smart Contracts', in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, IEEE, 2019, pp. 83–92. doi: 10.1109/EMPDP.2019.8671637.
- [27] R. Gelashvili *et al.*, 'Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing', in *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, in PPOPP '23. Association for Computing Machinery, 2023, pp. 232–244. doi: 10.1145/3572848.3577524.
- [28] M. Piduguralla, S. Chakraborty, P. S. Anjana, and S. Peri, 'DAG-Based Efficient Parallel Scheduler for Blockchains: Hyperledger Sawtooth as a Case Study', in *Euro-Par 2023: Parallel Processing*, in *Lecture Notes in Computer Science*, vol. 14100. Springer, 2023, pp. 184–198. doi: 10.1007/978-3-031-39698-4_13.
- [29] M. Bartoletti, L. Galletta, and M. Murgia, 'A Theory of Transaction Parallelism in Blockchains', *Log. Methods Comput. Sci.*, vol. 17, no. 4, 2021, doi: 10.46298/lmcs-17(4:10)2021.