systems

MDPI

Article

A Decision-Centric IoT and Blockchain-Enabled Supply Chain Information System for Demand-Aware Routing: Architecture and Multi-Seed Evaluation

Feisal Hadi Masmali and Tahir Hakim *

Department of Management Information Systems, College of Business, Jazan University, Jazan 45142, Saudi Arabia; fmasmali@jazanu.edu.sa
* Correspondence: thakim@jazanu.edu.sa

Highlights

Please indicate how your work links to systems science via your contributions to systems practice, theory, and/or methodology.

- Treats the supply chain information system as a closed decision loop in which every plan is validated against the partners' shared constraints, recorded, and repaired or rejected before execution, so constraint safety is a property of the loop rather than of the decision engine.
- Contributes a reproducible systems-evaluation method grounded in real multi-plant distribution data: 30-seed paired tests against published routing methods and single-component ablations, stress tests of every simulation assumption, and released code.

What are the main findings and/or the implications of the main findings?

- Pricing delivered demand in the routing objective lowers cost per unit by 33–45%; the learned engine is the fastest high-quality decision maker in the loop, and seeding search with its plan gives the lowest cost of every method tested, including ALNS and OR-Tools.
- Evidence-based validation kept every invalid plan from execution with up to four of five validators faulty, and validate-and-repair kept cost within 0.4% of fault-free operation at a 10% fault rate.

Abstract

Supply chain information systems increasingly need to support decisions, yet Internet of Things (IoT) and blockchain approaches remain largely data-centric. We propose a decision-centric system in which IoT data acquisition, an artificial intelligence decision engine and a blockchain layer form one closed loop: each daily routing plan is optimized jointly for cost, lateness and delivered demand, validated against shared partner constraints and recorded before execution. We evaluated it by simulation on the supply chain logistics problem dataset (41 real customers) over 30 paired seeds. A learned deep Q-network engine with decision-time rollout reduced cost per 1000 units delivered by 24.3% relative to a demand-aware heuristic and 28.0% relative to OR-Tools, deciding in 43 ms; seeding adaptive large neighborhood search (ALNS) with its plan gave the lowest cost, 28.0% below the heuristic and below ALNS alone in 29 of 30 seeds. Joint routing–demand optimization lowered cost per unit by 33–45% relative to a cost-only objective. Under hard time windows, a heterogeneous fleet and eleven other scenarios, rollout stayed 12–32% cheaper than the heuristic in all 130 paired runs, and seeded ALNS was cheapest in every setting. Evidence-based validation blocked every invalid plan with up to four of five validators faulty.

Check for updates

Academic Editor: Jianhua Zhu

Received: 21 September 2026
Revised: 27 October 2026
Accepted: 28 October 2026
Published: 2 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

Keywords: supply chain information systems; decision-centric systems; Internet of Things; blockchain; demand-aware routing; deep reinforcement learning; adaptive large neighborhood search; Byzantine fault tolerance

1. Introduction

Existing artificial intelligence (AI)-based approaches to logistics have largely split supply chain management into smaller, disconnected sub-problems. Modern supply chains do not follow the predictable, linear routes of classical models; they are complex, dynamic networks with continuous data generation, fluctuating demand, and operational uncertainty. Supply chain management is consequently moving toward integrated system modeling, optimization, and decision intelligence, in which routing and demand decisions are made jointly, in real time, and validated against a shared set of constraints through a blockchain-based governance mechanism.

Conventional supply chain information systems (SCIS) were designed mainly for data collection, monitoring, and reporting rather than for autonomous decision-making, and they become less useful under disruptive or fast-changing conditions. Supply chains therefore need systems that go beyond collecting and transforming data and act on it directly to streamline logistics operations. Figure 1 shows the unified decision engine proposed in this work, which integrates the Internet of Things (IoT), AI, and blockchain.

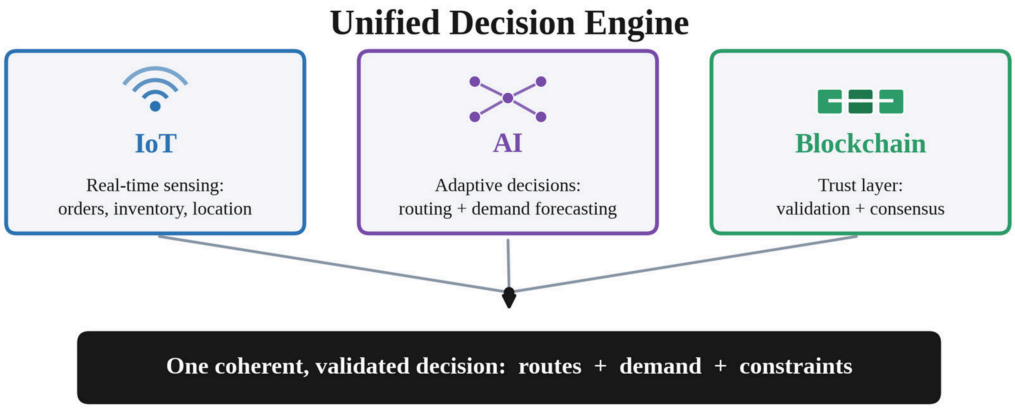


Figure 1. Decision-centric loop: IoT data feed the AI decision engine, whose plans are validated on the blockchain before execution.

IoT gives supply chains continuous, real-time visibility into asset location, condition, and status across the network, a key enabler of informed decision-making [1]. Most IoT-enabled supply chains, however, remain observational rather than prescriptive: they monitor conditions but do not act on them. Blockchain is highly effective at establishing trust and security among heterogeneous partners and at keeping an immutable record of transactions, but it rarely triggers or enforces operational decisions directly and works as a record of data rather than as a check on decisions. Artificial intelligence has separately advanced route planning and demand forecasting, yet these capabilities are usually treated as independent problems rather than as parts of one coherent decision process [2].

We therefore present a decision-centric system that integrates IoT, AI, and blockchain into a single, adaptive, and trustworthy engine for logistics decision-making. Section 1.2 states the research gap and the contributions.

1.1. Background and Motivation

Supply chains involve many interdependent stakeholders, including manufacturers, distributors, logistics providers, and retailers, that must coordinate despite fluctuating demand, unpredictable schedules, and limited resources [3]. Digital technologies, and intelligent systems in particular, increasingly serve as the backbone for coordinating these operations.

IoT provides real-time visibility into asset location, condition, and operational status, but visibility alone has limited value unless a downstream engine turns it into better decisions. Blockchain strengthens trust, security, and transparency among supply chain partners [4]. Although it establishes accountability, blockchain implementations to date rarely execute or validate decisions automatically.

Artificial intelligence, including machine learning and optimization algorithms, offers effective tools for routing and forecasting, but these tools are usually applied in isolation from one another. A routing plan that looks optimal under current conditions can quickly become suboptimal after an unanticipated demand spike.

Our objective is therefore to design a decision-centric system that links IoT data, AI-driven decision-making, and blockchain-based governance, so that logistics operations become adaptive, reliable, and trustworthy. The following subsection defines the research gap and summarizes the paper's contributions.

1.2. Contributions and Paper Organization

Three gaps remain in the literature reviewed in Section 2. First, integrated IoT–AI–blockchain frameworks use the ledger to secure or trace data, so a decision that violates the partners' operating constraints is recorded faithfully but not prevented. Second, the AI components of these frameworks optimize forecasting, queueing or routing in isolation rather than the joint decision of which customers to serve and how to route them when capacity is scarce. Third, learned logistics policies are rarely compared on real data with published search methods, so it is unclear when a learned decision engine should be preferred to optimization. From a systems perspective, the missing element is a feedback loop in which sensing, decision, validation and execution are designed and evaluated as one system, with explicit guarantees about what the loop will not execute.

The paper contributes to systems practice and methodology in four ways. First, it proposes a decision-centric supply chain information system in which validation is a gate inside the decision loop: every plan is checked against the partners' shared constraints, recorded, and repaired or rejected before execution, so that constraint safety is a property of the loop rather than of the decision engine. An evidence-based validation rule, introduced here, requires one honest validator instead of an honest majority. Second, it formulates the daily decision as a prize-collecting capacitated vehicle routing problem whose objective equals the undiscounted return of the sequential process solved by the learned routers, so learned and search-based engines are interchangeable behind one interface, and it shows that integrating demand into the objective lowers cost per unit by 33–45%, the largest effect of any architectural component (the learned engine without demand information lowered it by 12% relative to reactive dispatch, and validation leaves valid plans unchanged). Third, it shows that the learned engine, on its own at short budgets and strengthened by decision-time rollout at longer ones, achieves lower cost than ALNS at every search budget from 0.05 s to 0.1 s on 41 customers, 0.02 s to 1 s on 80 customers and 0.05 s to 2 s on 160 customers (mean decision time); with rollout it is 24.3% cheaper per unit than a demand-aware heuristic and 28.0% cheaper than OR-Tools in 43 ms, while seeding adaptive large neighborhood search (ALNS) with the learned plan gives the lowest cost when more time is available (28.0% below the heuristic and below ALNS alone in 29

of 30 seeds). Fourth, it provides a reproducible evaluation method grounded in real multi-plant distribution data: 30-seed paired tests against published routing methods and single-component ablations, stress tests of every simulation assumption, routing constraint, coverage policy and validator-network size from 4 to 100 validators, and released code that regenerates every number.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 presents the system model and problem formulation (Section 3.1), the architecture (Section 3.2) and the experimental setup (Section 3.3). Section 4 reports the results, Section 5 discusses their implications for systems design, management and sustainability together with the limitations, and Section 6 concludes the paper.

2. Related Work

As supply chains digitalize, IoT, blockchain, and AI have each received considerable research attention, both independently and in combination. Much of this work targets tracking, traceability, and forecasting; comparatively few studies integrate these technologies into a unified, trustworthy decision-making system, and this paper addresses that gap.

2.1. IoT-Enabled Supply Chain Systems

IoT gives supply chains a continuous, real-time feed of operational data, including asset location, condition, and status [5]. Prior studies attribute shorter delivery and lead times, better inventory control, and support for vehicle and warehouse maintenance to this increased transparency. IoT alone is often insufficient, however: knowing that a shipment is delayed does not indicate the appropriate corrective action. Most existing systems lack effective mechanisms for turning high-velocity streaming data into timely decisions; the data volume can overwhelm conventional processing pipelines, and device security and interoperability add further challenges. Real-time visibility is therefore necessary but not sufficient, and without decision support its value is only partly realized. A systematic review of the IoT-supply chain literature is consistent with this shortfall: it finds that most published work remains conceptual or descriptive and that empirical, quantitative evidence linking IoT adoption to measurable supply chain performance is still comparatively scarce [5]. Domain-specific studies report similarly strong but narrowly scoped gains from IoT-based sensing. An intelligent-warehousing system combining IoT with machine learning reports 99% accuracy but limited adaptability to new operating conditions [6], and an embedded-systems framework combines adaptive routing, IoT-enabled demand forecasting, and blockchain supply chain security [7], but applies the ledger to securing supply chain data rather than to validating routing decisions before they are executed.

2.2. Blockchain-Based Supply Chain Systems

Blockchain technology addresses trust concerns among supply chain partners with differing incentives. Its immutable, distributed record lets every participant verify shared information independently [8]. Reported applications include product tracking, counterfeit prevention, regulatory compliance, and process automation through smart contracts, with permissioned participants sharing a consistent, cryptographically secured view of the ledger [4]. In most implementations, however, blockchain serves as a secure record-keeping mechanism rather than a decision-making component, and it rarely extends to making or enforcing operational decisions under the real-time conditions typical of logistics. Scalability to large, high-throughput networks is also a significant challenge: empirical benchmarks of a permissioned blockchain platform show that transaction latency rises with both workload and the number of participating nodes, even as throughput stays stable for a fixed node count, confirming that consensus overhead is a measurable cost rather than a

purely theoretical one [9]. Blockchain alone is therefore insufficient to transform logistics decision-making. The same pattern recurs across domain-specific blockchain-IoT deployments that report security, trust, or efficiency gains in pharmaceuticals [10], healthcare [11], general logistics [12,13], construction and building-information-modeling contexts [14], and warehouse-location decisions [15], as well as in supply chain management platforms more broadly [16]; in each case, blockchain remains confined to record verification rather than becoming part of an active decision-making process. A related strand emphasizes privacy rather than decision support, proposing cryptographic approaches that strengthen data confidentiality at the cost of added latency [17].

2.3. AI for Routing and Demand-Aware Logistics

Artificial intelligence has substantially advanced logistics planning and forecasting. Machine learning, deep learning, evolutionary algorithms, and reinforcement learning have been applied to complex problems such as vehicle routing and demand forecasting [3]. In routing, AI-based search methods identify near-optimal paths; in forecasting, learned models estimate future demand from historical and contextual data. Reinforcement learning in particular has been proposed as an alternative to classical search-based routing. A recent deep reinforcement learning formulation of the dynamic and uncertain vehicle routing problem models routing as a partially observed Markov decision process and uses an attention-based network to adapt to real-time demand fluctuations, reporting gains over static routing heuristics under dynamic conditions [18]. For forecasting, federated learning has let supply chain partners predict delivery risk collaboratively without pooling raw order data; it improves prediction for organizations with smaller historical datasets, although the benefit is uneven, since firms with rich order histories still predict better on their own [19]. These techniques have become increasingly accurate and better able to handle complex, non-stationary patterns, improving inventory levels and delivery performance.

Routing and demand forecasting are nevertheless typically solved as separate problems, despite their interdependence: current demand influences routing decisions, and routing decisions in turn affect future demand patterns. Treating them independently, as most existing research does, limits overall system performance. Learned routing heuristics based on attention models approach the solution quality of specialized algorithms on benchmark vehicle routing instances [20], but they solve static instances, without live IoT data or governance of the resulting decisions.

Applied studies show how AI is moving from forecasting into operational supply chain decisions. An AI framework for recycling dormant and obsolete inventory, for example, turns inventory data into disposition decisions for stock that would otherwise be written off [21]. Learned routing policies are one such operational decision, but classical search remains a strong reference for them: adaptive large neighborhood search [22] and guided local search as implemented in OR-Tools [23] are standard tools for rich vehicle routing problems [24], and when capacity is insufficient to serve all customers the problem becomes a prize-collecting, orienteering-type problem in which the planner must also choose whom to serve [25]. A learned router therefore has to be compared with such search methods, not only with constructive heuristics, which this paper does in Section 4.1.

2.4. Integrated and Emerging Approaches

Fewer studies attempt to integrate all three technologies at once. One recent framework combines IoT edge controllers, blockchain-based transaction logging, and an AI-based optimizer to manage service-request queues in smart-city logistics, reporting improvements in queue length, resource utilization, and end-to-end delay [26]. Unlike the present work, it optimizes queue scheduling rather than joint vehicle routing and demand-aware

decision-making, and its blockchain layer is not evaluated as a source of decision-validation latency. A related but distinct line of work models the supply chain itself as a real-time, IoT-fed digital twin, a simulation-and-optimization layer used to anticipate and mitigate disruption risk and improve structural resilience [27]. This approach shares our emphasis on continuously feeding IoT data into a live decision model, but it does not incorporate blockchain-based validation or reinforcement-learning-based routing. Across this body of work, IoT, AI, and blockchain are more often combined pairwise, or integrated around a different decision objective such as queueing or disruption resilience, than around the joint routing-and-demand decision problem with blockchain-validated execution addressed here. Several reviews and conceptual studies survey this convergence without implementing or evaluating an integrated system of their own. A web-content-mining-based review of IoT decision-support systems in supply chains and logistics offers useful decision insights but no real-time execution [28]; broader reviews of blockchain-IoT supply chain applications report positive impact and improved flexibility without implementation or experimentation [29,30]; and a review focused on blockchain-IoT-AI convergence reports improved traceability but does not arrive at a unified model [31]. A conceptual account of supply chain digitization likewise reports improved resilience but stops short of incorporating AI [32]. Among empirical studies that combine more than one of the three technologies, a machine-learning-and-blockchain system improves prediction accuracy without adding a decision layer [33], and a vaccine supply chain system integrating machine learning, IoT, and blockchain improves forecasting at the cost of increased system complexity [34]. Together, these reviews and multi-technology studies reinforce that the specific combination pursued here, joint routing-and-demand decision-making validated through blockchain, remains largely unaddressed.

Motivated by this gap, we propose a decision-centric system that jointly integrates IoT, AI, and blockchain to support coordinated supply chain decision-making. Table 1 summarizes the relevant literature and the specific gaps this paper addresses; the problem is formalized in Section 3.1.

Table 1. Summary of the existing literature, with methods, results, and research gaps.

Ref.	Title	Specification	Year	Methodology	Results/Accuracy	Research Gap/Implications
[1]	IoT logistics analysis	Research trends	2024	Bibliometric	Trend analysis	No model
[2]	Blockchain IoT decision model	Decision model	2023	MCDM	Better selection	Static model
[3]	AI Supply Chain Optimization	Improves forecasting	2025	AI/ML	Better forecasting	No trust system
[4]	Food IoT Blockchain	Improves transparency	2020	DL + IoT	Improved safety	Scalability issue
[5]	Internet of Things and Supply Chain Management: A Literature Review	Broad IoT-SC review	2019	Systematic literature review	Maps IoT use across SC functions	Few empirical/quantitative studies
[6]	Intelligent Warehousing	99% accuracy	2025	ML + IoT	99% accuracy	Limited adaptability
[7]	The Potential of Embedded Systems in Optimizing Industrial Logistics	Adaptive routing, IoT demand forecasting, blockchain security	2024	Embedded IoT framework	Reported efficiency and cost gains	Ledger secures data; decisions not validated before execution
[8]	Smart logistics IoT Blockchain	Secure logistics	2020	Framework	Secure system	No optimization

Table 1. *Cont.*

Ref.	Title	Specification	Year	Methodology	Results/Accuracy	Research Gap/Implications
[9]	Empirical Performance Analysis of Hyperledger LTS for Small and Medium Enterprises	Blockchain performance benchmark	2022	Empirical benchmarking (up to 20 nodes)	Latency rises with workload and node count	No SC-specific workload; no decision layer
[10]	Secure pharmaceutical supply chain using blockchain in IoT cloud systems	Improves security	2024	IoT + Blockchain	Secure system	Scalability issue
[11]	Medical supply chain blockchain IoT	Improves tracking	2023	IoT + Blockchain	Improved trust	No optimization
[12]	IoT–Blockchain SC	Improves automation	2021	IoT + Blockchain	Better integration	No intelligence
[13]	IoT Blockchain logistics	Improves efficiency	2022	IoT + Blockchain	Better efficiency	Cost issue
[14]	IoT BIM Blockchain	Improves traceability	2024	Integration	Better trust	Domain specific
[15]	Warehouse optimization	Improves cost	2022	IoT + Blockchain	Cost reduction	Adaptability
[16]	IoT Blockchain SCM	Improves quality	2022	Model	Better security	No optimization
[17]	Privacy blockchain IoT	Improves privacy	2024	Crypto model	Secure system	Latency
[18]	Deep Reinforcement Learning for the Dynamic and Uncertain Vehicle Routing Problem	RL-based routing	2023	Deep RL (POMDP + attention network)	Outperforms static heuristics under dynamic demand	No blockchain validation; routing only
[19]	Federated Machine Learning for Privacy-Preserving, Collective Supply Chain Risk Prediction	Federated learning forecasting	2023	Federated learning across buyers	Improves prediction for data-poor firms	No routing/decision execution
[20]	Attention, Learn to Solve Routing Problems!	Learned routing heuristic	2019	Deep RL (attention model, REINFORCE)	Near specialized algorithms on TSP/VRP benchmarks	Static instances; no IoT data or decision governance
[26]	IoT and Blockchain-Based Secure Supply Chain Framework Using Optimal Queue Model	Integrated IoT + Blockchain + AI	2024	IoT edge + blockchain + AI queue optimizer	Improved queue length, utilization, and delay	Queue scheduling only, not joint routing–demand
[27]	A Digital Supply Chain Twin for Managing Disruption Risks and Resilience	Digital twin	2021	Real-time IoT-fed simula-tion/optimization	Improved disruption anticipation and resilience	No blockchain validation; no RL-based routing
[28]	A Review of Decision Support Systems in the Internet of Things and Supply Chain and Logistics Using Web Content Mining	IoT Adopter II improves decision-making	2024	Review + Web Mining	Better decision insights	No real-time execution
[29]	Blockchain IoT SCM review	Improves visibility	2021	Review	Better flexibility	No implementation
[30]	Blockchain IoT SC review	Improves sustainability	2023	Review	Positive impact	No experiment
[31]	Blockchain-IoT-AI review	Improves adaptability	2024	Review	Better traceability	No unified model
[32]	Supply chain digitization	Improves efficiency	2024	Conceptual	Better resilience	No AI integration
[33]	Blockchained supply chain management	Improves prediction	2022	ML + Blockchain	Better prediction	No decision layer

Table 1. *Cont.*

Ref.	Title	Specification	Year	Methodology	Results/Accuracy	Research Gap/Implications
[34]	Vaccine SC system	Improves trust	2023	ML + IoT + Blockchain	Better forecasting	Complexity
[35]	Blockchain IoT roadmap	Adoption factors	2023	Framework	Strategic insight	No implementation

AI, artificial intelligence; BIM, building information modeling; DL, deep learning; IoT, Internet of Things; MCDM, multi-criteria decision-making; ML, machine learning; POMDP, partially observable Markov decision process; RL, reinforcement learning; SC, supply chain; SCM, supply chain management; TSP, traveling salesman problem; VRP, vehicle routing problem.

Table 2 compares the integrated frameworks closest to this work on the properties that define a decision-centric system. The frameworks differ from the present one in the role given to the blockchain and in the decision that is optimized. In refs. [7,33,34] the ledger secures or traces data while the AI component forecasts or predicts; in ref. [26] the AI component schedules service queues; and the digital twin of [27] and the learned routers of [18,20] optimize decisions without a validation layer. The theoretical contribution of the present architecture lies in this combination: the decision itself is the object that is optimized jointly over routing and demand, validated against constraints agreed among the partners, recorded, and repaired or replaced before execution, so that constraint safety becomes a property of the closed loop rather than of any single decision engine.

Table 2. Comparison with integrated frameworks on the properties of a decision-centric system.

Ref.	IoT Input	AI Decision	Role of Blockchain	Decision Validated Before Execution	Joint Routing–Demand Objective
[7]	Yes	Adaptive routing; demand forecasting	Secures supply chain data	No	Not reported
[26]	Yes (edge)	Queue scheduling	Transaction logging	No	No
[27]	Yes	Simulation-optimization (digital twin)	None	No	No
[33]	Yes (tracking)	Prediction	Secures shared data	No	No
[34]	Yes	Demand forecasting	Secures shared data	No	No
[18]	No	Learned routing under dynamic demand	None	No	Partly (dynamic demand)
[20]	No	Learned routing on static instances	None	No	No
This work	Yes (simulated stream)	Joint routing and demand-aware service selection (learned or search engine)	Validates, records, and repairs or rejects each plan	Yes	Yes

3. Materials and Methods

This section formalizes the system and the decision problem (Section 3.1), describes the architecture (Section 3.2), and details the experimental setup (Section 3.3).

3.1. System Model and Problem Formulation

This section presents a formal mathematical framework for the proposed decision-centric supply chain information system. The methodology combines system modeling, optimization theory, and decision intelligence for dynamic, uncertain environments, and centers on a unified decision function that jointly optimizes routing and demand-aware logistics under blockchain-based validation constraints. Figure 2 gives a graphical overview of the end-to-end workflow of the study.

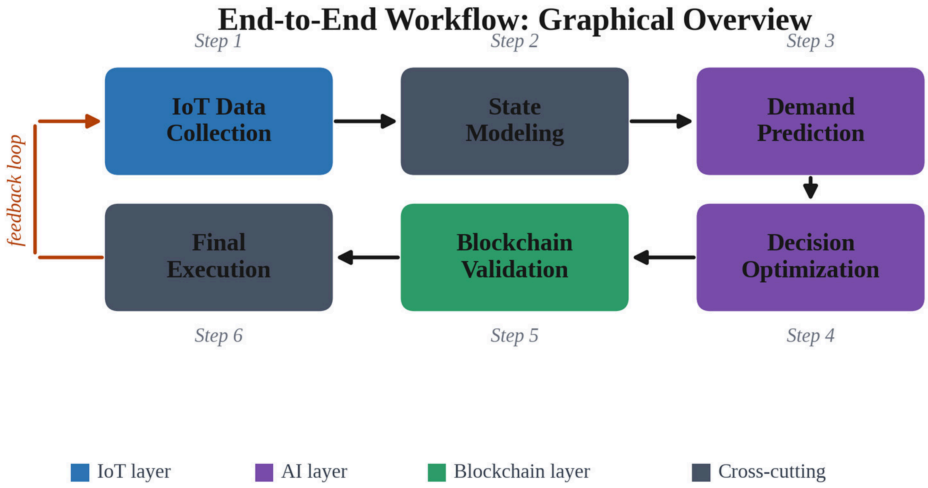


Figure 2. Workflow of the study: data, state, decision, validation, execution and feedback.

Table 3 lists the notation used in the paper.

Table 3. Notation.

Symbol	Meaning
$G = (V, E)$	Distribution network; $V = \{0\} \cup N$, depot 0, customers $N = \{1, \ldots, n\}$
D_{ij}	Road distance between nodes i and j (km)
K, k	Set of vehicle trips per day and trip index
$Q_k, \overline{Q}$	Capacity of trip k; mean trip capacity
κ_k	Carrier of trip k (drawn at the depot's real usage shares)
$F_\kappa(W)$	Real weight-banded freight charge of carrier κ for consolidated weight W
c_{km}	Local delivery cost per km
v, s, H	Vehicle speed, service time per stop, shift length
w_i, δ_i	Unit weight and due time of customer i
$d_{it}, \tilde{d}_{it}, \hat{d}_{it}$	True, IoT-observed and forecast demand of customer i on day t
$\sigma, p_{\mathrm{m}}, \lambda$	Observation-noise SD, missing-reading probability, EWMA smoothing factor
a^k_{ij}, y_{ik}	Arc and service indicators of trip k
q_{ik}, τ_{ik}	Planned load and arrival time of customer i on trip k
Z, J_{t}	Daily objective (Equation (7)); J_{t} for day t
$\alpha_1, \ldots, \alpha_4$	Weights of cost, lateness, delivered demand and coverage
S_t, A_t	System state and daily plan (decision)
s, a, r	State, action and reward of the trip-construction process
γ, ε	Discount factor and exploration rate
$\mathcal{C}_t$	Set of plans that satisfy the shared constraints on day t
$V(A)$	Validation function
N_v, f	Number of validators and number of faulty validators
β, τ_T	Trust smoothing factor and trust threshold
ω_j	Violation witness attached to validator j's reject vote

We model the AI-driven supply chain optimization framework as a directed graph that captures the relationships among warehouses, distribution centers, and customers through interconnected transportation links. The model includes dynamic state representation, IoT-based data streams, and stochastic transition dynamics to reflect real-world uncertainties. Bringing demand, cost, location, and resource availability into a unified state space allows the framework to support intelligent and adaptive decision-making for efficient logistics operations.

3.1.1. System Model

Let the supply chain be represented as a directed graph:

$$G = (V,E) \tag{1}$$

where

- $V = \{v_1, v_2, \ldots, v_n\}$ represents nodes (warehouses, distribution centers, customers)
- $E = \{e_{ij}\}$ represents edges (transportation links)

State Representation

The system state at time t is defined as:

$$S_t = \{L_t, D_t, C_t, R_t\} \tag{2}$$

where

- $L_t = \{l_i(t)\}$: location states
- $D_t = \{d_i(t)\}$: demand vector
- $C_t = \{c_{ij}(t)\}$: transportation cost matrix
- $R_t = \{r_k(t)\}$: resource availability

IoT Data Stream Model

$$\begin{aligned} X_t &= \{x_1(t), x_2(t), \ldots, x_m(t)\} \\ x_i(t) &= f_i(sensor, location, time) \\ S_t &= f(X_t) \end{aligned} \tag{3}$$

Transition Dynamics

$$S_{t+1} = T(S_t, A_t, \omega_t) \tag{4}$$

where

- A_t: decision/action
- ω_t: stochastic disturbance

3.1.2. Problem Formulation

Define the decision variable:

$$A_t = \left\{ a_{ij}^k(t) \right\} \tag{5}$$

where

$$a_{ij}^k(t) = \begin{cases} 1, & \text{if vehicle } k \text{ travels from } i \to j \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Total trip capacity is smaller than total demand (Section 3.3.1), so not every customer can be served on a given day and the planner must also choose which customers to serve. The routing problem is therefore a prize-collecting capacitated vehicle routing problem with a route-duration limit and soft due times, of the team-orienteering family [24,25]. In addition to the arc variables $a^k{}_{ij}$, we use the service indicator $y_{ik} \in \{0, 1\}$, the planned load $q_{ik} \geq 0$ and the arrival time τ_{ik} of customer i on trip k.

Objective Function

Multi-objective optimization:

$$\min Z = \alpha_1 Z_1 + \alpha_2 Z_2 + \alpha_3 Z_3 - \alpha_4 \sum_k \sum_i y_{ik} \tag{7}$$

(1) Routing Cost

$$Z_1 = \sum_k \left[c_{\mathrm{km}} \sum_i \sum_j D_{ij}\, a_{ij}^k + F_{\kappa_k} \left(\sum_i w_i\, q_{ik} \right) \right] \tag{8}$$

(2) Time Delay

$$Z_2 = \sum_k \sum_i (\tau_{ik} - \delta_i)^+ y_{ik} \tag{9}$$

(3) Delivered Demand

$$Z_3 = -\frac{1}{\overline{Q}}\sum_k \sum_i q_{ik} \tag{10}$$

Z_1 is the freight charge of each trip's consolidated weight under its carrier's real weight-banded rates plus the local distance cost, Z_2 the lateness of the stops made, and Z_3 the demand actually delivered, which replaces the forecast-error term of the initial formulation; α_4 is an optional coverage weight (Section 4.6). The weights are $\alpha_1 = 0.1$, $\alpha_2 = 0.5$, $\alpha_3 = 3.0$ and $\alpha_4 = 0$, and J_t denotes Z evaluated for day t.

Constraints

Flow Conservation:

$$\sum_j a_{ij}^k = \sum_j a_{ji}^k = y_{ik} \quad \forall i \in N,\ k; \qquad \sum_{j \in N} a_{0j}^k \le 1\ \forall k \tag{11}$$

Capacity Constraint:

$$\sum_i q_{ik} \le Q_k, \qquad 0 \le q_{ik} \le \tilde{d}_{it}\, y_{ik} \qquad \forall i,k \tag{12}$$

Single Visit:

$$\sum_k y_{ik} \le 1 \quad \forall i \tag{13}$$

Time Propagation and Subtour Elimination:

$$\tau_{jk} \ge \tau_{ik} + s\,[i \ne 0] + \frac{D_{ij}}{v} - M\left(1 - a_{ij}^k\right) \qquad \forall i,j,k,\ \tau_{0k} = 0 \tag{14}$$

Route Duration:

$$\tau_{ik} + s + \frac{D_{i0}}{v} \le H + M(1 - y_{ik}) \qquad \forall i,k \tag{15}$$

Hard Time Windows (variant, Section 4.5):

$$\tau_{ik} \le \delta_i + M(1 - y_{ik}) \qquad \forall i,k \tag{16}$$

Binary Constraint:

$$a_{ij}^k,\ y_{ik} \in \{0,1\} \tag{17}$$

Constraint (13) allows at most one visit per customer and day; it replaces the requirement of the initial formulation that every customer be visited, which cannot hold when capacity binds. Loads may be partial (12), and (14) also eliminates subtours. Plans are built from observed demand $\tilde{d}_{it}$ and executed against true demand.

3.1.3. Decision-Centric Framework

The decision pipeline is defined as:

$$X_t \to \hat{S}_t \to \hat{D}_t \to A_t \tag{18}$$

Prediction Function

$$\hat{D}_t = f_\theta(X_t) \tag{19}$$

Decision Function

$$A_t = \pi\left(S_t, \hat{D}_t\right) \tag{20}$$

Policy Optimization

$$\pi^* = \underset{\pi}{\text{argmin}} \mathrm{E}\left[\sum_{t=0}^{T} Z(S_t, A_t)\right] \tag{21}$$

3.1.4. AI-Driven Decision Engine

Routing Optimization (RL Formulation)

$$Q(s,a) = \mathrm{E}\left[r_t + \gamma \underset{a'}{\max} Q(s', a')\right] \tag{22}$$

Policy (ε-greedy in training, greedy at test)

$$\pi(a \mid s) = \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)|, & a = \arg\max\limits_{a' \in \mathcal{A}(s)} Q(s, a') \\ \varepsilon/|\mathcal{A}(s)|, & \text{otherwise} \end{cases} \tag{23}$$

Admissible Actions

$$\mathcal{A}(s) = \left\{ j \in U : \ \widetilde{d}_{jt} > 0, \ \ t + \frac{D_{ij}}{v} + s + \frac{D_{j0}}{v} \le H \left[t + \frac{D_{ij}}{v} \le \delta_j \right] \right\} \tag{24}$$

where $s = (k, i, t, c, U, \widetilde{d})$ is the state of the trip being built (current node, elapsed time, remaining capacity and unvisited customers), so constraints (11)–(17) hold by construction for every plan the learned routers produce.

Demand Forecasting Model

$$\hat{d}_i(t+1) = f_{ML}(d_i(t), d_i(t-1), \ldots, x_i(t)) \tag{25}$$

Integrated Decision Model

$$A_t^* = \underset{A_t}{\text{argmin}} Z\left(S_t, \hat{D}_t, A_t\right) \tag{26}$$

3.1.5. Optimization Model

Multi-Objective Formulation

$$\text{min} Z = (Z_1, Z_2, Z_3) \tag{27}$$

Pareto Optimality

$$A^* \in \{A : \nexists A' \text{ such that } Z(A') \prec Z(A)\} \tag{28}$$

Weighted Sum Method

$$Z = \sum_{i=1}^{3} \alpha_i Z_i \tag{29}$$

Lagrangian Formulation

$$\mathcal{L} = Z + \lambda_1(\mathit{Capacity}) + \lambda_2(\mathit{Flow}) \tag{30}$$

Stochastic Optimization

$$\text{min} \mathrm{E}_\omega[Z(S_t, A_t, \omega)] \tag{31}$$

Constraint Relaxation

$$\text{min} Z + \mu \sum(\text{violations}) \tag{32}$$

To make explicit which formulations are solved in the experiments: the daily problem (7)–(17) is solved by every decision engine, directly by ALNS and through the equivalent

sequential process (22)–(24) by the learned routers, and the validation model (33)–(39) is implemented as stated. The pipeline and policy formulations (18)–(21) and (26) describe the decision problem as a whole, which the learned routers optimize in expectation; the generic forecasting model (25) is instantiated by the exponentially weighted forecast (78); and the multi-objective, Lagrangian, stochastic and constraint-relaxation forms (27)–(32) state the general problem class and are not solved directly. Likewise, in Section 3.3, Equations (66)–(76) and (79)–(81) state the general data, sensing and forecasting models, of which the evaluation uses the observation model (77)–(78), and (102)–(106) characterize the baselines schematically.

3.1.6. Blockchain-Based Validation Model

Decision Record

$$B_t = H(A_t \| S_t \| t) \tag{33}$$

where H is a cryptographic hash function.

Validation Function

$$V(A_t) = \begin{cases} 1, & A_t \in \mathcal{C} \\ 0, & \text{otherwise} \end{cases} \tag{34}$$

Smart Contract Constraint

$$A_t \in \mathcal{C} \tag{35}$$

Consensus Mechanism

$$A_t^{final} = \text{Consensus}\left(A_t^{(1)}, A_t^{(2)}, \ldots, A_t^{(n)}\right) \tag{36}$$

Trust Score

$$\text{Trust}_i(t) = \beta\, \text{Trust}_i(t-1) + (1-\beta)\, \mathbb{1}[v_i(t) = \text{maj}(t)] \tag{37}$$

Because membership of $\mathcal{C}$ is decidable from the plan itself, a reject vote can carry a violation witness ω_j = (constraint, trip, stop) that any node can re-check. The evidence rule introduced in this revision rejects a plan whenever at least one witness verifies, ignores reject votes whose witness does not verify, and scores validators against the verified outcome $\hat{V}(A_t)$ instead of the majority:

$$\text{accept}(A_t) \Leftrightarrow \nexists j : \ \text{Verify}(\omega_j, A_t), \ \ \text{Trust}_j(t) = \beta\, \text{Trust}_j(t-1) + (1-\beta)\, \mathbb{1}\left[V_j(A_t) = \hat{V}(A_t)\right] \tag{38}$$

Safety then requires one honest validator rather than an honest majority. A rejected plan is replaced by the reactive dispatch plan or, in the validate-and-repair mode, minimally repaired (phantom and repeated stops removed, loads clipped, trips truncated to return within the shift), so the corrected execution rule is

Final Decision Execution

$$A_t^{\text{exec}} = \begin{cases} A_t, & \text{accept}(A_t) \\ \mathcal{R}(A_t) \text{ or } A_t^{\text{dispatch}}, & \text{otherwise} \end{cases} \tag{39}$$

Trust score is defined here as a per-validator exponential moving average of agreement with the consensus outcome, so it is conceptually distinct from the consensus mechanism above: the consensus mechanism determines whether one specific decision is accepted

in a single round, whereas trust score tracks a validator's reliability across many rounds. Figure 3 shows the decision framework as implemented in the evaluation, tracing how real-time data move through the system from raw observations to executed routing decisions.

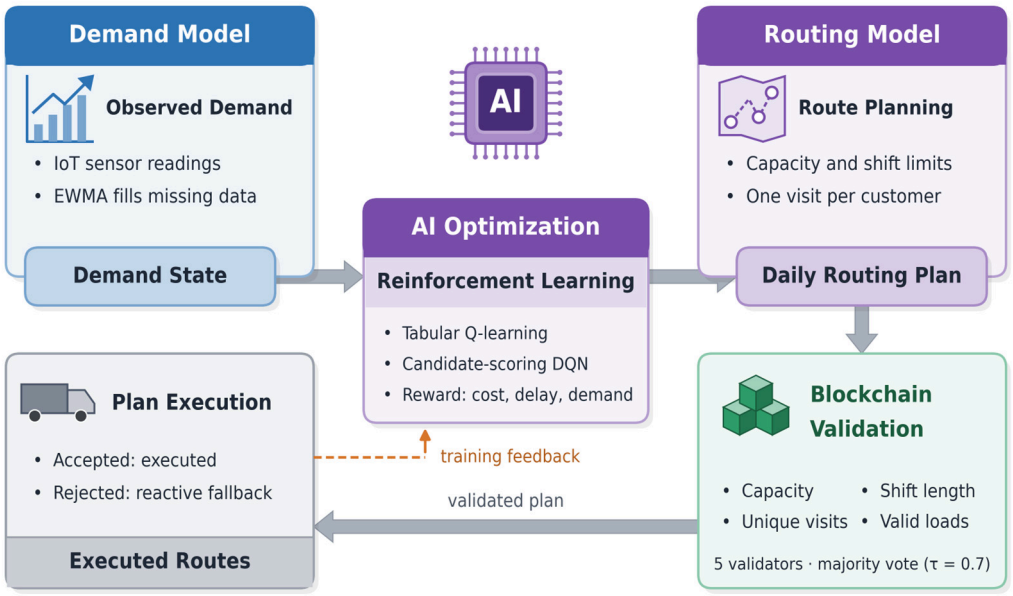


Figure 3. Decision framework as implemented: IoT-observed demand, AI routing, blockchain validation of each daily plan, execution or fallback, and training feedback.

In the framework of Figure 3, the IoT data layer supplies each customer's observed demand, with missing readings filled by an exponentially weighted moving-average forecast. The AI routing engine, a tabular Q-learning router or a candidate-scoring deep Q-network trained on a joint cost, delay, and delivered-demand reward, turns this state into a daily routing plan. Five validators check the plan against the shared constraints (capacity, shift length, one visit per customer, valid customer identifiers, and loads within observed demand) and accept it by majority vote among validators whose trust score is at least the threshold; every decision is recorded in a SHA-256 hash chain. An accepted plan is executed against true demand, a rejected plan is repaired or replaced by the reactive dispatch plan, and the executed outcomes update the router during training (Section 3.3).

3.2. Proposed Decision-Centric SCIS Architecture

The proposed architecture puts a decision-centric paradigm into operation by integrating IoT-based data acquisition, AI-driven decision intelligence, and blockchain-based governance in one system. Unlike traditional data-centric architectures, it treats decision-making as its core function, supporting real-time, adaptive, and trustworthy logistics operations. A layered design provides modularity, scalability, and interoperability across heterogeneous supply chain environments.

3.2.1. Overall Architecture

The overall architecture is structured into four primary layers. Figure 4 illustrates a blockchain-enabled decision-validation framework in which optimized decisions are verified through smart contracts and consensus algorithms. Decision nodes process and validate outcomes based on cost, delay, and trust, and the blockchain ledger provides secure, immutable, and distributed record-keeping.

$$\mathcal{A} = \{L_{IoT}, L_{Data}, L_{Decision}, L_{Blockchain}\} \tag{40}$$

Mathematical Flow

$$X_t \rightarrow S_t \rightarrow \hat{D}_t \rightarrow A_t \rightarrow V(A_t) \rightarrow A_t^{exec} \tag{41}$$

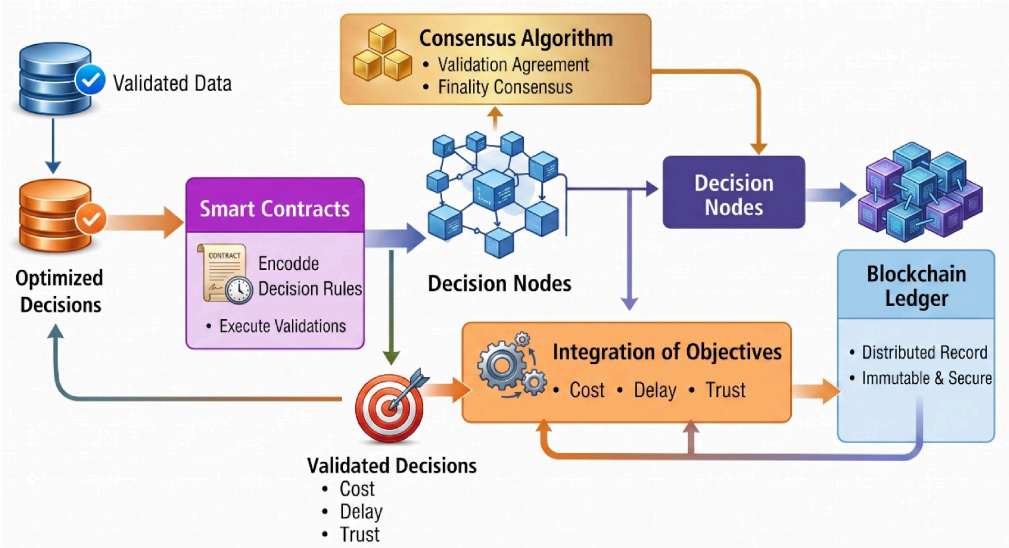


Figure 4. Decision validation: each plan is checked against the shared constraints, recorded, and released or rejected.

Each layer performs a specialized function within the decision-centric workflow, which keeps concerns separated while the system components remain integrated.

3.2.2. IoT Layer: Data Acquisition and Streaming

The IoT layer is the foundation of the architecture and acquires data in real time from distributed supply chain entities. It consists of interconnected devices such as sensors, global positioning system (GPS) trackers, radio-frequency identification (RFID) tags, and embedded systems that continuously monitor logistics operations.

Data Model

$$\begin{aligned} X_t &= \{x_1(t), x_2(t), \ldots, x_n(t)\} \\ x_i(t) &= (l_i, temp_i, status_i, time_i) \end{aligned} \tag{42}$$

Streaming Function

$$X_t = \phi(Sensor, Network, Time) \tag{43}$$

The captured data include vehicle locations, shipment conditions, environmental parameters, and inventory states, and this real-time stream supports continuous monitoring and situational awareness.

To handle high-velocity data, streaming frameworks are employed:

$$X_t^{processed} = \psi(X_t) \tag{44}$$

where ψ represents filtering, aggregation, and preprocessing operations.

Challenges Modeled

$$\begin{aligned} &Noise(X_t),\ Delay(X_t),\ Missing(X_t) \\ &X_t^{clean} = f_{clean}(X_t) \end{aligned} \tag{45}$$

The IoT layer thereby keeps an up-to-date representation of the supply chain state for downstream decision-making.

3.2.3. Decision Intelligence Layer: AI Engine Integration

The decision intelligence layer is the core of the proposed architecture: it turns processed data into optimized decisions using machine learning models, reinforcement learning policies, and optimization techniques.

State Input

$$S_t = f(X_t) \tag{46}$$

Demand Prediction

$$\hat{D}_t = f_\theta(S_t) \tag{47}$$

Decision Function

$$A_t = \pi\left(S_t, \hat{D}_t\right) \tag{48}$$

Optimization Objective

$$A_t^* = \underset{A_t}{\operatorname{argmin}} Z\left(S_t, \hat{D}_t, A_t\right) \tag{49}$$

This layer jointly optimizes routing and demand-aware logistics to capture their interdependencies, with reinforcement learning models adapting routing strategies dynamically:

$$Q(s,a) \rightarrow \pi(a|s) \tag{50}$$

Feedback Loop

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{51}$$

Combining predictive and prescriptive analytics lets the system respond to real-time changes such as demand fluctuations and traffic conditions.

The decision layer hosts interchangeable engines behind the same interface: the learned routers, which decide in milliseconds and can spend additional time on decision-time rollout, and an adaptive large neighborhood search (ALNS) engine that optimizes (7) directly and can start from the learned engine's plan (Section 3.3.3). Because all engines optimize the same objective and their plans pass through the same validation, the engine can be chosen by decision cadence without changing the rest of the system.

3.2.4. Blockchain Governance Layer: Smart Contracts and Validation

The blockchain layer ensures trust, transparency, and accountability in decision-making, and unlike traditional implementations, it governs decisions rather than only data.

Decision Encoding

$$B_t = H(A_t \| S_t \| t) \tag{52}$$

Validation Function

$$V(A_t) = \begin{cases} 1, & A_t \in \mathcal{C} \\ 0, & \text{otherwise} \end{cases} \tag{53}$$

Consensus Mechanism

$$A_t^{final} = \text{Consensus}\left(A_t^{(1)}, \ldots, A_t^{(n)}\right) \tag{54}$$

Trust Score

$$\text{Trust}_i(t) = \beta\,\text{Trust}_i(t-1) + (1-\beta)\,\mathbb{1}[v_i(t) = \text{maj}(t)] \tag{55}$$

Execution Rule

$$A_t^{\text{exec}} = \begin{cases} A_t, & \text{accept}(A_t) \\ \mathcal{R}(A_t) \text{ or } A_t^{\text{dispatch}}, & \text{otherwise} \end{cases} \tag{56}$$

Smart contracts enforce constraints such as capacity limits, delivery deadlines, and stakeholder agreements:

$$A_t \in \mathcal{C} \tag{57}$$

Blockchain thereby becomes an active decision-governance mechanism rather than a passive ledger.

In a deployment the validators would be operated by the supply chain partners and reach agreement through a Byzantine fault-tolerant protocol; Section 4.9 quantifies its latency and message cost for 4 to 100 validators. The shared constraints are the operational terms the partners agree on, so the ledger acts as an executable agreement on what a valid decision is.

3.2.5. Decision Workflow: End-to-End System Operation

Figure 5 shows how the complete workflow integrates all layers into a continuous decision-making pipeline.

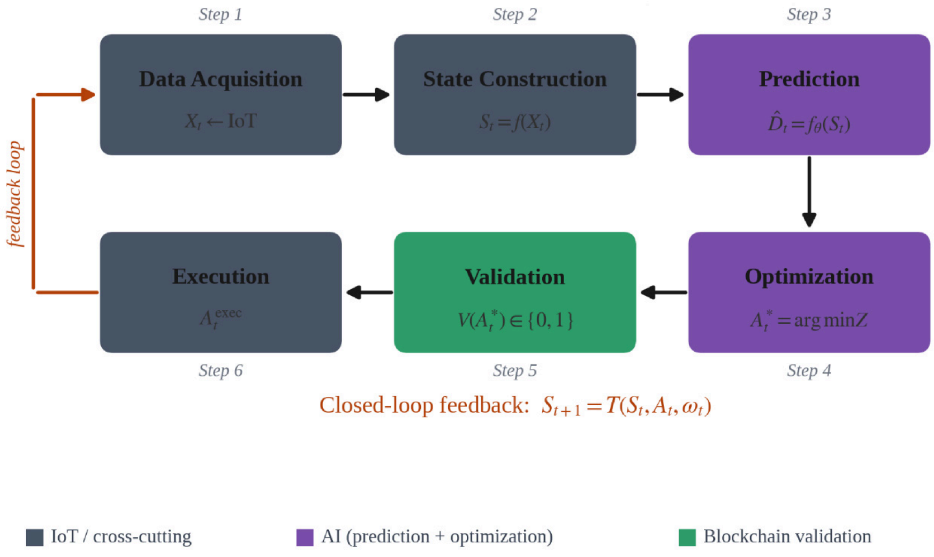


Figure 5. End-to-end decision workflow: data acquisition, state construction, prediction, optimization, validation, and execution, with closed-loop feedback.

Stepwise Flow

Step 1: Data Acquisition

$$X_t \leftarrow IoT \tag{58}$$

Step 2: State Construction

$$S_t = f(X_t) \tag{59}$$

Step 3: Prediction

$$\hat{D}_t = f_\theta(S_t) \tag{60}$$

Step 4: Optimization

$$A_t^* = \text{argmin} Z \tag{61}$$

Step 5: Validation

$$V(A_t^*) \rightarrow Blockchain \tag{62}$$

Step 6: Execution

$$A_t^{exec} \tag{63}$$

Closed-Loop Feedback

$$S_{t+1} = T(S_t, A_t, \omega_t) \tag{64}$$

Workflow Equation

$$A_t^{exec} = \mathcal{G}(X_t, \theta, \pi, \mathcal{C}) \tag{65}$$

This closed loop ensures continuous adaptation and learning, allowing the supply chain to respond dynamically to changing conditions.

3.3. Experimental Setup

This section describes the experimental design we used to evaluate the proposed decision-centric supply chain information system. The setup combines real-world datasets, simulated IoT data streams, AI-based decision models, and blockchain-based validation mechanisms, with the aim of rigorously assessing system performance under realistic, dynamic logistics conditions.

This section describes how the system was evaluated: the real data that ground the simulation (Section 3.3.1), the simulated IoT streams (Section 3.3.2), the decision engines and their training (Section 3.3.3), the ledger implementation (Section 3.3.4), and the metrics and baselines (Section 3.3.5). Table 4 separates the components taken from real data from those that are simulated or assumed, and states where the sensitivity of the results to each assumption is tested.

Table 4. Real, assumed and simulated components of the evaluation.

Component	Source	Basis	Sensitivity Tested
Customers, order quantities, unit weights, service levels	Real (OrderList, PLANT03)	41 real customers; bootstrap of each customer's own orders	Demand shift +30% (Section 4.7); 10–160 customers (Section 4.8)
Carriers, carrier shares, freight rates	Real (FreightRates, OrderList)	Weight-banded tables, PORT04→PORT09	Cost per km ×0.5–×4 (Section 4.7)
Customer locations	Assumed	No coordinates in the data; uniform in 100 × 100 km, new layout per seed	Clustered layout; 200 × 200 km (Section 4.7); 30 layouts
Fleet and capacity	Assumed	3 trips/day; capacity 0.55 × mean daily demand	Capacity 0.40–1.00; 2 and 5 trips; heterogeneous fleet (Sections 4.5 and 4.7)
Speed, service time, shift, due times	Assumed	40 km/h; 0.25 h; 8 h; 3/5/8 h by service level	Due times ×0.67; hard windows (Sections 4.5 and 4.7)
IoT demand stream	Simulated	5% relative noise, 2% missing, EWMA fill	Noise 0–40%, missing 0–10% (Section 4.7)
Ledger and validators	Simulated	SHA-256 chain, 5 validators, measured CPU time	PBFT with 4–100 validators over LAN/WAN; faulty validators (Sections 4.9 and 4.10)

3.3.1. Dataset Description

The experimental evaluation uses the supply chain logistics problem dataset (Kalganova & Dzalbs, Brunel University London, 2019) [36], a real historical multi-plant distribution network with real plant-to-port connections, real per-plant daily capacities, a real

weight-banded freight-rate schedule per carrier, and 9215 real historical orders. How we map this dataset onto the system's routing/demand/cost interface is detailed below; the dataset was introduced with the ant colony optimization study of [37].

The real dataset used in Section 4 comprises seven relational tables: OrderList (9215 real historical orders, each with a plant code, customer ID, order date, product ID, unit quantity, and shipment weight), FreightRates (a weight-banded rate schedule per carrier and port-pair, giving a minimum charge and a per-unit-weight rate for each weight bracket), PlantPorts (which plants connect to which ports), WhCapacities (each plant's real daily throughput capacity), WhCosts, ProductsPerPlant, and VmiCustomers. This evaluation uses the network's dominant plant, PLANT03, as the depot: it accounts for 8541 of the dataset's 9215 orders (92.7%) and serves 41 of the dataset's 46 distinct real customers, so almost the entire real order volume and customer base is represented.

Each of the 41 real customers' empirical demand distributions is built directly from that customer's own historical order quantities, bootstrap-resampled once per simulated day and winsorized at the 95th percentile of PLANT03's order quantities (12,331 units) to control a small number of extreme bulk orders without discarding the customers who place them. Each customer's unit weight is the median ratio of shipment weight to unit quantity over its orders, and its service level (door-to-door, DTD; door-to-port, DTP; or customer-referred freight, CRF) is its most frequent level. Three vehicle trips are dispatched per day, and the per-trip capacity is set to 0.55 of the mean total daily demand divided by the number of trips (22,133 units), so capacity binds and not all demand can be served. Each trip's carrier is drawn at PLANT03's real usage shares (V444_0: 68.4%; V444_1: 23.2%; V44_3: 8.3%). Its freight charge is computed from that carrier's real weight-banded rate table for the PORT04 to PORT09 lane (DTP rates for V444_0 and DTD rates for V444_1, matching how each carrier is used in the order history) as the maximum of the minimum charge and rate times consolidated trip weight, taking the cheapest rate row where several exist and never charging more than the next weight bracket would. Carrier V44_3 carries only CRF, so its trips incur no freight charge to the shipper. Because the source data contain no customer coordinates, customers are placed uniformly at random in a 100 km by 100 km region with the depot at its center, with a new layout for each seed. Vehicles travel at 40 km/h with 0.25 h of service per stop within an 8 h shift; due times are 3 h (DTD), 5 h (DTP), and 8 h (CRF) after dispatch; and local delivery costs 0.02 cost units per km. These assumed parameters are listed in Table 5 and in the released code, and their influence is tested in Section 4.7. Local delivery is a substantial share of daily cost, so routing decisions affect both which demand is served and what it costs.

Table 5. Key real quantities extracted from the dataset for the simulation.

Quantity	Description
Depot Plant	PLANT03—92.7% of the dataset's 9215 historical orders (8541); serves 41 of 46 distinct real customers
Depot Daily Capacity	1013 (WhCapacities; recorded for reference). Per-trip capacity used: 0.55 × mean total daily demand/3 trips = 22,133 units
Carrier Usage Share	V444_0: 68.4% (DTP rates); V444_1: 23.2% (DTD rates); V44_3: 8.3% (CRF, no charge to the shipper); drawn per trip
Demand Winsorization Cap	12,331 units—95th percentile of PLANT03 order quantities (all customers pooled)
Assumed operating parameters	3 trips/day; 40 km/h; 0.25 h service per stop; 8 h shift; due 3 h (DTD), 5 h (DTP), 8 h (CRF); local cost 0.02 per km
Assumed IoT parameters	Observation noise N(0, 0.05^2) relative; 2% missing readings, filled by EWMA forecast (smoothing 0.3)

CRF, customer-referred freight; DTD, door-to-door; DTP, door-to-port; EWMA, exponentially weighted moving average.

No classification or regression labels are used from this dataset; it grounds the simulation's routing graph, demand, capacity, and cost structure rather than serving as a supervised-learning training set. The 800/200 epoch training/held-out test split used in Section 4 applies to the simulated decision epochs the routing policies are trained and evaluated on, not to a split of the historical orders themselves. Table 5 summarizes the real quantities extracted from the dataset for the simulation, and Table 6 summarizes the key fields used from its source tables.

Table 6. Key fields of the supply chain logistics problem dataset.

Category	Field	Description
Order Identification	Order Date	Date each historical order was placed (OrderList)
Network Topology	Plant Code, Customer	Real depot plant and real customer identifiers linking each order (OrderList)
Network Topology	Origin Port, Destination Port	Real ports connecting the depot plant to the customer network (OrderList, PlantPorts)
Demand	Product ID	Identifies the product shipped on each order (OrderList)
Demand	Unit quantity	Real historical order quantity in units, resampled per simulated epoch (OrderList)
Demand	Weight	Real shipment weight, used to compute freight cost (OrderList)
Capacity	Plant ID	Identifies which plant a capacity record belongs to (WhCapacities)
Capacity	Daily Capacity	Real per-plant daily throughput capacity (WhCapacities)
Freight Cost	Carrier	Real carrier identifier, e.g., V444_0, V444_1 (FreightRates)
Freight Cost	minm_wgh_qty, max_wgh_qty	Weight-bracket bounds for a freight-rate tier (FreightRates)
Freight Cost	rate, minimum cost	Per-unit-weight rate and minimum charge for a weight bracket (FreightRates)
Freight Cost	tpt_day_cnt	Transit-time-in-days tier used to select a freight-rate row (FreightRates)
Service Level	Service Level	Carrier service tier, e.g., CRF, DTD (OrderList)
Service Level	Ship Ahead/Late Day Count	Days a shipment was sent ahead of or behind schedule (OrderList)
Warehouse Cost	Warehouse Cost	Per-unit handling cost at a warehouse (WhCosts)
Product Coverage	Products Per Plant	Which products a given plant produces (ProductsPerPlant)
Customer Coverage	VMI Customers	Which customers a given plant serves under vendor-managed inventory (VmiCustomers)
Disclosed Simplification	Local-Delivery Layout (synthetic)	Synthetic per-customer coordinates, uniform in a 100 km × 100 km region with a new layout per seed; not present in the source data
Network Scale	Number of Real Customers	41 distinct real customers served by the depot plant, out of 46 in the full dataset
Network Scale	Total Historical Orders	9215 real historical orders across all plants; 8541 (92.7%) from the depot plant used here
Network Scale	Winsorization Cap	12,331 units—95th percentile of PLANT03 order quantities (all customers pooled)

VMI, vendor-managed inventory.

Dataset Representation

Let the dataset be defined as:

$$\mathcal{D} = \left\{ (l_i, d_i, t_i, c_{ij}) \right\}_{i=1}^{N} \tag{66}$$

where

- l_i: location coordinates
- d_i: demand values
- t_i: timestamps
- c_{ij}: transportation cost between nodes

Dataset Integration

$$\mathcal{D}_{combined} = \mathcal{D}_1 \cup \mathcal{D}_2 \tag{67}$$

Preprocessing

$$\begin{aligned} \hat{d}_i &= \frac{d_i - \mu_d}{\sigma_d} \\ \hat{c}_{ij} &= \frac{c_{ij}}{\max(c)} \end{aligned} \tag{68}$$

Temporal Segmentation

$$\mathcal{D}_{train}, \mathcal{D}_{test} = split(\mathcal{D}, \rho) \tag{69}$$

where

$$\rho = 0.8 \tag{70}$$

The datasets provide information on shipments, orders, transportation costs, and delivery timelines, enabling realistic modeling of supply chain operations.

3.3.2. IoT Data Simulation

Because real-time IoT streams are not directly available, we developed a simulation framework to emulate dynamic data generation.

IoT Stream Model

$$\begin{aligned} X_t &= \{x_i(t)\} \\ x_i(t) &= (l_i(t), v_i(t), status_i(t), temp_i(t)) \end{aligned} \tag{71}$$

Location Update

$$l_i(t+1) = l_i(t) + v_i(t)\Delta t + \varepsilon_t \tag{72}$$

Noise Model

$$\varepsilon_t \sim \mathcal{N}(0, \sigma^2) \tag{73}$$

Delay Model

$$x_i^{delayed}(t) = x_i(t - \delta) \tag{74}$$

Missing Data Simulation

$$P(x_i(t) = \emptyset) = p_m \tag{75}$$

Data Cleaning

$$X_t^{clean} = f_{interp}(X_t) \tag{76}$$

In the evaluation, each customer's observed demand is its true demand multiplied by $(1 + \varepsilon)$, with $\varepsilon \sim N(0, 0.05^2)$ and truncation at zero, and a reading is missing with probability 0.02, in which case it is replaced by an exponentially weighted moving-average (EWMA) forecast (smoothing factor 0.3) of that customer's past observed demand. Routing plans

are built from observed demand and executed against true demand, so observation error propagates into delivered quantities and cost. The sensitivity analysis (Section 4.7) varies the noise level from 0% to 40%.

Formally, the observed demand and the forecast used to fill missing readings are

$$\tilde{d}_{it} = \begin{cases} d_{it} \max(0,\, 1+\varepsilon_{it}), & \text{with probability } 1-p_m,\ \varepsilon_{it} \sim \mathcal{N}(0, \sigma^2) \\ \hat{d}_{it}, & \text{otherwise} \end{cases} \tag{77}$$

$$\hat{d}_{i,t+1} = (1-\lambda)\, \hat{d}_{it} + \lambda\, \tilde{d}_{it} \tag{78}$$

3.3.3. AI Model Implementation

The decision intelligence layer consists of two core components: demand prediction and routing optimization. Two learned routers instantiate the routing component. Both build each vehicle trip one stop at a time from the customers that are unvisited, have positive observed demand, and can be reached and returned from within the shift; a trip ends when its capacity is exhausted or no customer remains feasible.

Demand Prediction Model

$$\hat{d}_i(t+1) = f_\theta(d_i(t), x_i(t)) \tag{79}$$

Loss Function

$$\mathcal{L} = \sum_i \left(d_i(t) - \hat{d}_i(t) \right)^2 \tag{80}$$

Evaluation Metrics

$$\begin{aligned} RMSE &= \sqrt{\tfrac{1}{N} \textstyle\sum \left(d_i - \hat{d}_i \right)^2} \\ MAE &= \tfrac{1}{N} \textstyle\sum \left| d_i - \hat{d}_i \right| \end{aligned} \tag{81}$$

Routing Optimization (RL-based)

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max Q(s',a') - Q(s,a)] \tag{82}$$

The neural router minimizes the Huber temporal-difference loss with a target network θ^- over the replay buffer $\mathcal{B}$:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{B}} \left[\ell_{\mathrm{H}} \left(r + \gamma \max_{a' \in \mathcal{A}(s')} Q_{\theta^-}(s', a') - Q_\theta(s,a) \right) \right] \tag{83}$$

Reward Function

$$\begin{aligned} r(s,a) = -\alpha_1 \left[c_{\mathrm{km}} D_{ia} + F_{\kappa_k}(W + w_a q_a) - F_{\kappa_k}(W) \right] - \alpha_2 (\tau_a - \delta_a)^+ + \alpha_3 \tfrac{q_a}{Q} \\ + \alpha_4 \end{aligned} \tag{84}$$

Policy

$$\pi(s) = \arg\max_{a \in \mathcal{A}(s)} Q(s,a) \ (\text{test}) \tag{85}$$

Integrated Decision Model

$$A_t^* = \mathrm{argmin} Z(S_t, \hat{D}_t, A_t) \tag{86}$$

The first router is tabular Q-learning. Its state encodes the vehicle's current position together with a five-bin discretization of the fraction of total observed demand not yet served, and its action is the next customer to visit. The identity of the specific unvisited customers is therefore not part of the state.

The second router is a deep Q-network with a shared candidate-scoring network. For each feasible customer, a ten-dimensional feature vector is formed: distance from the current position, distance back to the depot, slack to the customer's due time, remaining capacity and shift time, share of customers still unvisited, local density of unvisited customers, the customer's observed demand, the load it would receive, and the marginal freight-plus-distance cost of serving it. A multilayer perceptron (MLP) with two hidden layers of 32 rectified linear (ReLU) units scores each vector, and the highest-scoring customer is chosen. Because the candidate set itself encodes which customers remain unvisited, and the same network scores every candidate, the router has 1441 parameters regardless of network size. It is trained with experience replay (buffer 20,000, batch 64), a target network synchronized every 500 updates, a Huber loss, the Adam optimizer (learning rate 10^{-3}), and an exploration rate that decays linearly from 1.0 to 0.05 over the first 60% of training.

Both routers receive a per-stop reward $r = -\alpha_1 \cdot \Delta\text{cost} - \alpha_2 \cdot \text{delay} + \alpha_3 \cdot \text{delivered}/Q$, where Δcost is the marginal freight-plus-distance cost of the stop, delay is its lateness in hours, delivered is the quantity actually delivered, and Q is the vehicle capacity ($\alpha_1 = 0.1$, $\alpha_2 = 0.5$, $\alpha_3 = 3.0$). This is the per-step form of the objective in Section 3.1.2: rewarding delivered quantity at each stop is equivalent, up to a constant per day, to penalizing unmet demand. The separate configurations use the cost term alone and receive no demand information: the tabular separate state replaces the remaining-demand fraction with the fraction of customers unvisited, and the neural separate router has its demand and load features set to zero and its marginal-cost feature limited to distance. Both routers are trained for 800 simulated days and evaluated, without exploration or learning, on 200 held-out days.

The freight increments in (84) telescope to the trip's freight charge, and the return leg is charged at the last stop, so the undiscounted return ($\gamma = 1$) of a day's plan equals $-J_t$, and maximizing it is the same problem as minimizing the expected daily objective (7). The routers are trained with $\gamma = 0.9$, which weights the stops of a day unequally and therefore optimizes a discounted surrogate of (7) with lower-variance bootstrapped targets. The discount factor affects training only: every reported result is the undiscounted objective (7) and its metrics evaluated on the executed plans, and the rollout and ALNS engines evaluate (7) exactly. With $\gamma = 0.99$, the setting closest to the undiscounted objective, the router still undercut the demand-aware heuristic in every seed (Section 4.4). Because rewards are computed from the executed outcome, the routers learn to minimize the objective under observation noise.

Table 7 summarizes the states, actions, objectives and settings of the decision engines.

Training follows the same loop for both routers. On each of 800 simulated days, a day's demand and carriers are drawn, the IoT stream is generated, the router builds the plan with ε-greedy exploration, the plan is executed against true demand, the per-stop rewards are computed from the executed outcome, and the router is updated (one tabular update per stop, or one minibatch update of the neural router per stored transition after a 500-transition warm-up). Every 80 days the current policy is evaluated greedily on 20 validation days that are disjoint from the test days, which yields the learning diagnostics of Section 4.4. After training, exploration and learning are switched off, and the policy is evaluated on the 200 held-out test days.

The search engine is adaptive large neighborhood search (ALNS) [22], a standard metaheuristic for rich vehicle routing problems. It starts from the demand-aware greedy plan, evaluates every candidate plan with the exact daily objective (7) (real freight bands, partial loads, lateness and route-duration feasibility), and at each iteration removes 1 to 8 customers by a random, worst-contribution or relatedness operator and reinserts customers by greedy or regret-2 insertion. Candidate plans are accepted by simulated annealing,

operator weights adapt to their recent success, and each trip of the best plan is finally improved by 2-opt. It runs 150 iterations per day (about 1 s on one core) in the main experiment and is given wall-clock budgets from 0.02 to 2 s in the latency study.

Table 7. Decision engines: state, actions, objective and settings.

Item	Tabular Q-Learning	Neural (Candidate-Scoring DQN)	ALNS (Search)
State/input	(current node, 5-bin share of observed demand still unserved)	10 features per feasible customer: distance from current node and to depot, slack to due time, remaining capacity and shift, share unvisited, local density, observed demand, deliverable load, marginal freight-plus-distance cost	Complete plan; observed demand
Action/move	Next customer from $\mathcal{A}(s)$	Next customer from $\mathcal{A}(s)$ (highest score)	Destroy 1–8 customers (random, worst, related); repair by greedy or regret-2 insertion
Objective	Reward (84), $\gamma = 0.9$	Reward (84), $\gamma = 0.9$	Daily objective (7)
Model/size	Q-table (grows with n)	MLP 10–32–32–1, ReLU; 1441 parameters for any n	No learned parameters
Training/search	800 days; $\eta = 0.1$; $\varepsilon = 0.1$	800 days; Adam 10^{-3}; replay 20,000; batch 64; target sync 500 updates; Huber loss; ε 1.0→0.05 over 60% of training	150 iterations per day (≈1 s); simulated annealing; adaptive operator weights; final 2-opt
Decision-time improvement	—	Rollout over the $K = 4$ or 8 top-scoring candidates (87); 2-opt	Optionally seeded with the learned rollout plan
Test	200 held-out days, greedy	200 held-out days, greedy	200 held-out days

The learned router is further strengthened at decision time by rollout, a policy-improvement step in which the router simulates its own continuation. At each stop, the K admissible customers with the highest learned scores are tried in turn; for each, the remainder of the day's plan is completed with the learned router itself and scored with the exact objective (7) on observed demand, and the customer whose completed plan scores best is committed:

$$\begin{aligned} a^*(s) &= \arg\min_{a \in \mathcal{T}_K(s)} J_t\Big(\text{complete}_{\pi_\theta}\Big(A_t^{\text{prefix}} \oplus a\Big)\Big), \;\; \mathcal{T}_K(s) \\ &= \text{top}-K_{a \in \mathcal{A}(s)}\, Q_\theta(s,a) \end{aligned} \tag{87}$$

where $\mathcal{T}_K(s)$ is the set of the K highest-scoring admissible actions and $\text{complete}_{\pi_\theta}$ finishes the plan with the learned policy. With $K = 1$ the rule reduces to the learned router, and because the router's own choice is always among the candidates and every completion is deterministic, the planned objective J_t (on observed demand) of the rollout plan is never higher than that of the learned router's own plan. The resulting plans are finally improved by 2-opt. Rollout uses no additional training; $K = 4$ and $K = 8$ are evaluated. The learned engine can also seed search: ALNS started from the rollout plan instead of the demand-aware plan is evaluated with the same 150-iteration budget.

3.3.4. Blockchain Simulation and Implementation

We implemented a lightweight permissioned ledger to validate and govern decisions. Each day's routing plan, consisting of the ordered stops and planned loads of every trip, is a decision record. Five validators independently check it against the shared constraints (vehicle capacity, shift length including the return to the depot, at most one visit per customer per day, valid customer identifiers, and planned loads not exceeding observed demand) and vote. Under the majority rule, the plan is accepted if a majority of the

validators whose trust score is at least $\tau = 0.7$ approve it; under the evidence rule (38) it is rejected whenever a reject vote carries a verified witness. A rejected plan is replaced by the reactive dispatch plan or, in the validate-and-repair mode, by its repair (39). Trust is an exponential moving average ($\beta = 0.9$) of each validator's agreement with the outcome. Accepted and rejected records are appended to a SHA-256 hash chain.

Block Structure

$$B_t = (A_t, S_t, t, h_{t-1}) \tag{88}$$

Hash Function

$$h_t = H(A_t \| S_t \| t \| h_{t-1}) \tag{89}$$

Validation Constraint

$$A_t \in \mathcal{C} \tag{90}$$

Consensus Mechanism

$$A_t^{final} = \operatorname*{argmax}_a \sum_{i=1}^{N} \mathbb{1}\left[A_t^{(i)} = a\right] \tag{91}$$

Trust Score

$$\mathrm{Trust}_i(t) = \beta\, \mathrm{Trust}_i(t-1) + (1-\beta)\, \mathbb{1}[v_i(t) = \mathrm{maj}(t)] \tag{92}$$

Execution Condition

$$A_t^{\mathrm{exec}} = \begin{cases} A_t, & \mathrm{accept}(A_t) \\ \mathcal{R}(A_t) \text{ or } A_t^{\mathrm{dispatch}}, & \text{otherwise} \end{cases} \tag{93}$$

Latency Measurement

$$Latency = t_{confirmation} - t_{submission} \tag{94}$$

The consensus mechanism is the majority vote expressed above; a vote is used rather than an arithmetic mean because averaging the binary routing variables a_{ij}^k defined in Section 3.1.2 would in general violate the binary decision constraint. The feasible decision set is denoted C throughout the paper. Validation time (all validators' checks) and consensus time (vote tally, trust update, hashing, and appending) are measured as the wall-clock time of the implementation on a single CPU core. The ledger is a simulation of hashing, validation, and consensus, not a deployment on a platform such as Hyperledger Fabric or Ethereum; communication between validators is not modeled, so the measured times are a lower bound for a deployed network.

To quantify what a deployed validator network would cost, a discrete-event simulation of the Practical Byzantine Fault Tolerance (PBFT) normal case [38] (pre-prepare, prepare and commit phases with quorum $2f + 1$ of $N_v = 3f + 1$ validators) was run for $N_v = 4$ to 100. Every node has a sequential processing queue and a serialized uplink. Processing costs are measured on the evaluation machine: Ed25519 signing and verification, SHA-256 hashing and the constraint check of real daily plans. One-way message delays are log-normal ($\sigma = 0.3$) with medians of 0.2 ms (local network, 1 Gbit/s), 20 ms (regional wide-area network, 100 Mbit/s) and 80 ms (global wide-area network, 50 Mbit/s). Each configuration was run for 100 decisions with all validators responsive and with f validators silent.

3.3.5. Evaluation Metrics

The system is evaluated on seven metrics, computed per held-out day and averaged over the 200 test days of each seed. Total Cost is the day's freight plus local delivery cost; because serving more demand costs more, Cost per 1000 Units delivered is reported as the

primary efficiency measure. Demand Satisfaction is the delivered share of true demand. Delivery Delay is the mean lateness, in hours, of the stops made. Routing Efficiency is the ratio of each trip's length after 2-opt improvement of its own stop sequence to its actual length, so a value of 1 means the sequence cannot be shortened by 2-opt. Customers Served counts the stops made per day, a measure of coverage. Decision Latency is the measured wall-clock time to produce, and where applicable validate, the day's plan.

Two groups of measures were added in this revision. Coverage is measured per day (customers served) and over the horizon (share of customers served at least once, and Jain's fairness index of service frequencies), and vehicle-km per 1000 units delivered is reported as a proxy for transport emissions:

$$\mathrm{UC}_t = 1000\,\frac{\mathrm{Cost}_t}{\sum_i q_{it}^{\mathrm{exec}}}, \quad \mathrm{KPU}_t = 1000\,\frac{\sum\limits_k L_k}{\sum\limits_i q_{it}^{\mathrm{exec}}}, \quad \mathcal{J} = \frac{\left(\sum_i \phi_i\right)^2}{n\sum_i \phi_i^2} \tag{95}$$

where q_{it}^{exec} is the delivered quantity, L_k the length of trip k and φ_i the share of days on which customer i is served.

(1) Total Cost

$$\mathrm{Cost}_t = \sum_k \left[c_{\mathrm{km}} L_k + F_{\kappa_k}(W_k)\right] \tag{96}$$

(2) Delivery Time

$$Delay = \sum \max\left(0, T_{actual} - T_{expected}\right) \tag{97}$$

(3) Demand Satisfaction

$$DS = \frac{\sum\limits_i d_i^{served}}{\sum\limits_i d_i} \tag{98}$$

(4) Routing Efficiency

$$RE = \frac{L_{2-opt}(\mathrm{trip})}{L_{actual}(\mathrm{trip})} \tag{99}$$

(5) Decision Latency

$$DL = t_{decision} - t_{input} \tag{100}$$

(6) Blockchain Overhead

$$BO = \frac{t_{blockchain}}{t_{total}} \tag{101}$$

Baseline Methods

Two groups of baselines are used. The published routing methods, which use the same fleet, capacity and shift, are nearest-neighbor construction with 2-opt (NN + 2-opt), a demand-aware greedy construction that selects the feasible customer with the largest deliverable load per kilometer followed by 2-opt (Demand-aware + 2-opt), Google OR-Tools with guided local search [23] applied to the prize-collecting formulation (7)–(17) with unsplittable loads and freight linearized per carrier inside the solver, followed by the partial-load top-up used by the demand-aware construction, and a 1 s limit per day, and ALNS [22]. The architecture ablations are IoT-only and Blockchain-only (reactive nearest-neighbor dispatch, without and with validation), AI-only (tabular router without validation), the proposed system with a tabular router, and the separate (cost-only) variants of both routers.

Baseline 1: IoT-Only System

$$A_t = f(X_t) \tag{102}$$

(no optimization)

Baseline 2: AI-Only Optimization

$$A_t = \text{argmin} Z(S_t) \tag{103}$$

(no blockchain)

Baseline 3: Blockchain-Based Data System

$$B_t = H(X_t) \tag{104}$$

(no decision optimization)

Baseline 4: Separate Optimization

$$Z = Z_1 + Z_2 \tag{105}$$

(independent routing & demand)

Comparison Function

$$\Delta Z = Z_{baseline} - Z_{proposed} \tag{106}$$

Experimental Configuration

Hyperparameters

Tabular Q-learning: learning rate $\alpha = 0.1$, discount $\gamma = 0.9$, exploration $\varepsilon = 0.1$. Neural router: discount 0.9, Adam learning rate 10^{-3}, replay buffer 20,000, batch 64, target update every 500 updates, exploration decaying from 1.0 to 0.05. Reward weights: $\alpha_1 = 0.1$, $\alpha_2 = 0.5$, $\alpha_3 = 3.0$. Ledger: five validators, trust smoothing $\beta = 0.9$, trust threshold $\tau = 0.7$. These values follow common defaults; no hyperparameter search was performed, so the reported results do not depend on tuning against the test days; Table S1 (Supplementary Materials) reports twelve alternative settings.

Simulation Horizon

$$\begin{aligned} t &= 1, 2, \ldots, T \\ T &= 1000 \end{aligned} \tag{107}$$

Hardware Environment

- CPU only; Python 3.13.16 with NumPy 2.5.3, pandas 3.0.5, SciPy 1.18.1, openpyxl 3.1.5, Matplotlib 3.11.2, OR-Tools 9.15 and cryptography 50.0.1 (no GPU or deep-learning framework)
- Main experiment suite: about 15 min; search baselines, variant and validator-network studies: about 10 h on two cores

Reproducibility

30 independent seeds (s = 0, . . ., 29). Every random stream is derived from the seed, and re-running the released code reproduces every reported metric exactly, apart from wall-clock timings.

Complexity

$$O(m\,n\,h) \tag{108}$$

Here m is the number of stops in a day, n the number of customers and h the hidden width: the learned routers score at most n candidates per stop with an MLP of width h. Rollout multiplies this by K times the number of remaining stops, and an ALNS iteration costs $O(r{\cdot}n{\cdot}m)$ route evaluations for r removed customers. Validation and consensus costs are reported separately (Sections 4.9 and 4.10).

Software, Code Availability, and Use of Generative AI

All experiments were run with the software listed above. The complete simulation and analysis code, the per-seed result files, and the scripts that generate every table and figure are provided as Supplementary Material (Software S1). Generative AI was used during this study: Claude (Anthropic; Claude Opus 5) assisted in writing and debugging the simulation and analysis code, in checking reference metadata, and in editing the manuscript text. All reported results were produced by executing the released scripts.

4. Results

This section first compares the system with published routing methods on the same instances (Section 4.1) and then isolates the contribution of each architectural component (Sections 4.2 and 4.3). Sections 4.4–4.8 examine learning behavior, additional routing constraints, the coverage–cost trade-off, robustness to the simulation assumptions, and scalability together with the latency of each decision engine. Sections 4.9 and 4.10 evaluate the validation layer: its cost and scalability, and its ability to keep invalid decisions from being executed.

4.1. Comparison with Published Routing Methods

Table 8 compares the decision engines of the proposed system with published routing methods that use the same fleet, capacity, shift and objective: NN + 2-opt, demand-aware greedy + 2-opt, OR-Tools and ALNS. The learned engine with decision-time rollout ($K = 4$) reduced cost per 1000 units delivered by 24.3% relative to the demand-aware heuristic (30 of 30 seeds; Wilcoxon $p < 0.001$), by 28.0% relative to OR-Tools (30 of 30 seeds; $p < 0.001$) and by 44.7% relative to NN + 2-opt, at the same demand satisfaction (0.555) and while deciding in 43 ms, about 21 times faster than ALNS. Rollout improved the learned router itself by 10.5% in every seed ($p < 0.001$) without any additional training, and $K = 8$ lowered cost per unit to 0.201. Seeding ALNS with the learned rollout plan gave the lowest cost of all methods, 0.194 per 1000 units: 28.0% below the demand-aware heuristic, 31.5% below OR-Tools, and below ALNS started from the heuristic plan in 29 of 30 seeds ($p < 0.001$), with no late deliveries and a shorter run time (839 against 917 ms). The learned engine therefore improves the best search method when it seeds it and is the fastest high-quality engine on its own.

Without rollout, the learned router decides in 2.55 ms including validation and is 15.5% cheaper per unit than the demand-aware heuristic (29 of 30 seeds) and 19.6% cheaper than OR-Tools (30 of 30 seeds); this is the operating point for decisions needed within milliseconds. OR-Tools, which searches the prize-collecting formulation with unsplittable loads and linearized freight and then tops up each trip with partial loads, packed capacity with many small customers (19.3 stops per day) and incurred more distance and lateness than the demand-aware construction; the ALNS and rollout engines, which evaluate the exact objective with partial loads and banded freight, avoided this. The formulation differences are confined to OR-Tools' internal search: its routing solver accepts only additive arc and node costs and whole node demands, so the banded freight charge, a step function

of each trip's consolidated weight, cannot be expressed in its objective. Every engine's plan, OR-Tools' included, is executed against true demand and scored with the same exact objective and metrics, so Table 8 compares the engines on the cost that the system actually incurs. Section 4.8 places all engines on the quality–latency plane.

Table 8. Comparison with published routing methods (mean ± SD; 30 seeds, 200 held-out days each).

Method	Cost per 1000 Units	Demand Satisfaction	Delay (h)	Customers per Day	Vehicle-km per 1000 Units	Decision Latency (ms)	Cost per Unit vs. Demand-Aware, % (Seeds Better)
NN + 2-opt	0.368 ± 0.061	0.531 ± 0.024	0.260 ± 0.108	24.5	6.26	0.80	−36.9 (0/30)
Demand-aware + 2-opt	0.269 ± 0.027	0.555 ± 0.005	0.024 ± 0.011	8.9	4.77	0.28	—
OR-Tools GLS, 1 s	0.283 ± 0.014	0.553 ± 0.004	0.105 ± 0.031	19.3	6.99	1004.0	−5.1 (6/30)
ALNS, 150 iterations	0.195 ± 0.014	0.555 ± 0.005	0.000 ± 0.000	7.8	5.07	916.8	+27.4 (30/30)
Learned router (neural)	0.227 ± 0.015	0.554 ± 0.005	0.024 ± 0.008	8.4	5.96	2.55	+15.5 (29/30)
Learned router + 2-opt	0.224 ± 0.015	0.554 ± 0.005	0.028 ± 0.010	8.4	5.72	2.62	+16.6 (30/30)
Learned router + rollout, K = 4	0.203 ± 0.014	0.555 ± 0.004	0.001 ± 0.001	7.8	5.24	43.3	+24.3 (30/30)
Learned router + rollout, K = 8	0.201 ± 0.014	0.555 ± 0.004	0.001 ± 0.000	7.8	5.17	80.0	+25.2 (30/30)
ALNS seeded by learned rollout	0.194 ± 0.013	0.555 ± 0.005	0.000 ± 0.000	7.8	5.04	839.3	+28.0 (30/30)

ALNS, adaptive large neighborhood search; GLS, guided local search; NN, nearest neighbor. Latency includes validation for the ledger-enabled configurations.

4.2. Contribution of the Architectural Components

Table 9 isolates the architectural components. Adding learned, demand-aware decisions to reactive IoT-driven dispatch lowered cost per 1000 units from 0.384 to 0.311 with the tabular engine, 0.227 with the neural engine and 0.195 with ALNS. The Blockchain-only configuration produced exactly the routing outcomes of IoT-only, as expected for a layer that validates rather than generates decisions, and every ledger in every seed and configuration was verified as intact. Demand satisfaction lies between 0.53 and 0.56 for all non-separate configurations because three trips of 22,133 units cannot carry all 41 customers' demand; the demand-aware engines reach this ceiling with 7.8 to 11.0 stops per day instead of 24.5, which is the source of their efficiency and of the coverage trade-off analyzed in Section 4.6.

Table 10 tests the neural configuration against each baseline with paired Wilcoxon signed-rank tests, Holm-corrected across the 24 comparisons. It was better than the IoT-only, Blockchain-only, AI-only, tabular and NN + 2-opt configurations on all four metrics ($p < 0.001$), and against the demand-aware heuristic it lowered total cost by 15.8% and cost per unit by 15.5% at the same delivery delay ($p = 0.685$) and a demand satisfaction within 0.2 percentage points (0.554 versus 0.555). Routing efficiency shows that 2-opt would shorten the neural engine's trips by 3.5% and the tabular engine's by 9.3%, consistent with the gain from 2-opt post-optimization in Table 8. Figure 6 presents these results per metric and Figure 7 the relative differences.

Table 9. Architecture ablations and engines (mean ± SD over 30 seeds; 200 held-out days per seed).

Metric	IoT-Only	AI-Only (Tabular)	Blockchain-Only	Proposed, Tabular	Proposed, Neural	NN + 2-opt	Demand-Aware + 2-opt	Proposed, ALNS
Total cost per day	24.19 ± 3.61	19.70 ± 1.39	24.19 ± 3.61	19.70 ± 1.39	14.96 ± 0.99	23.16 ± 3.47	17.78 ± 1.77	12.90 ± 0.91
Cost per 1000 units	0.384 ± 0.065	0.311 ± 0.027	0.384 ± 0.065	0.311 ± 0.027	0.227 ± 0.015	0.368 ± 0.061	0.269 ± 0.027	0.195 ± 0.014
Demand satisfaction	0.531 ± 0.024	0.535 ± 0.009	0.531 ± 0.024	0.535 ± 0.009	0.554 ± 0.005	0.531 ± 0.024	0.555 ± 0.005	0.555 ± 0.005
Delivery delay (h)	0.213 ± 0.108	0.243 ± 0.100	0.213 ± 0.108	0.243 ± 0.100	0.024 ± 0.008	0.260 ± 0.108	0.024 ± 0.011	0.000 ± 0.000
Routing efficiency	0.958 ± 0.018	0.907 ± 0.022	0.958 ± 0.018	0.907 ± 0.022	0.965 ± 0.008	1.000 ± 0.000	1.000 ± 0.000	0.999 ± 0.001
Customers served per day	24.5 ± 2.3	11.0 ± 1.0	24.5 ± 2.3	11.0 ± 1.0	8.4 ± 0.4	24.5 ± 2.3	8.9 ± 0.4	7.8 ± 0.2
Vehicle-km per 1000 units	6.55	8.76	6.55	8.76	5.96	6.26	4.77	5.07
Customers served in 200 days (%)	88.5	78.8	88.5	78.8	87.6	88.5	95.9	82.9
Service fairness (Jain)	0.67	0.36	0.67	0.36	0.31	0.65	0.38	0.28
Decision latency (ms)	0.63 ± 0.15	0.53 ± 0.18	0.81 ± 0.20	0.68 ± 0.25	2.55 ± 0.66	0.80 ± 0.21	0.28 ± 0.09	917

SD, standard deviation; NN, nearest neighbor; ALNS, adaptive large neighborhood search. Rows without SD are means over seeds.

Table 10. Relative difference of the proposed system (neural engine) from each baseline (%; positive = proposed better; Holm-adjusted Wilcoxon p, 30 paired seeds).

Baseline	Total Cost	Cost per 1000 Units	Demand Satisfaction	Delivery Delay
IoT-only	+38.1 ($p < 0.001$)	+40.9 ($p < 0.001$)	+4.3 ($p < 0.001$)	+88.5 ($p < 0.001$)
Blockchain-only	+38.1 ($p < 0.001$)	+40.9 ($p < 0.001$)	+4.3 ($p < 0.001$)	+88.5 ($p < 0.001$)
AI-only (tabular)	+24.1 ($p < 0.001$)	+26.9 ($p < 0.001$)	+3.5 ($p < 0.001$)	+89.9 ($p < 0.001$)
Proposed, tabular	+24.1 ($p < 0.001$)	+26.9 ($p < 0.001$)	+3.5 ($p < 0.001$)	+89.9 ($p < 0.001$)
NN + 2-opt	+35.4 ($p < 0.001$)	+38.2 ($p < 0.001$)	+4.3 ($p < 0.001$)	+90.6 ($p < 0.001$)
Demand-aware + 2-opt	+15.8 ($p < 0.001$)	+15.5 ($p < 0.001$)	−0.3 ($p < 0.001$)	−2.3 ($p = 0.685$)

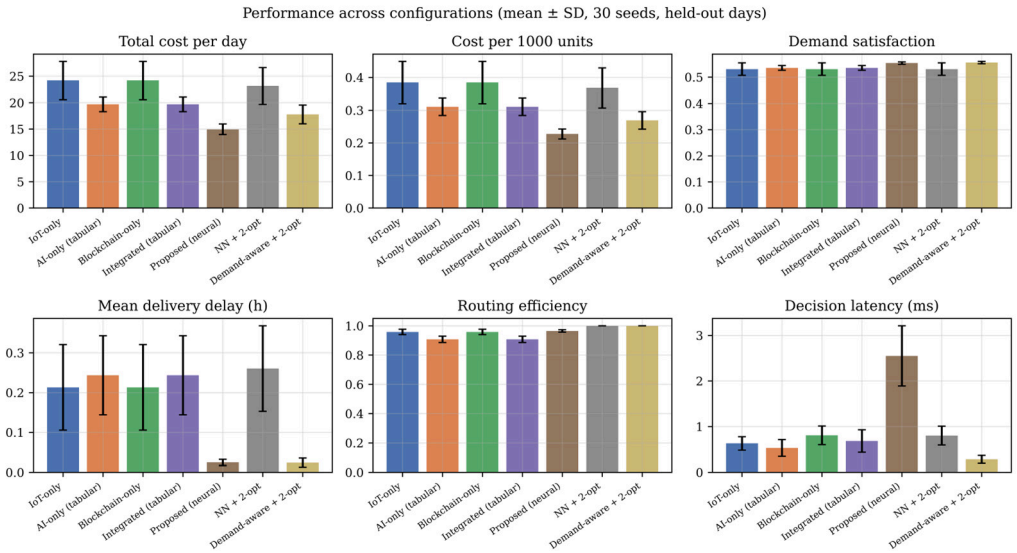


Figure 6. Performance of all configurations (mean ± SD over 30 seeds).

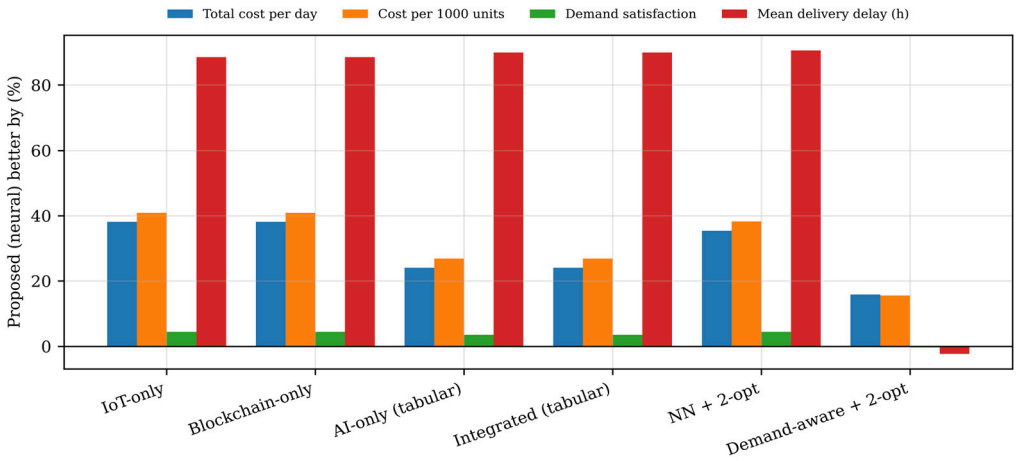


Figure 7. Relative difference of the proposed system from each baseline on four metrics (positive = proposed better; 30 seeds).

4.3. Impact of Decision Integration

Table 11 isolates the effect of integrating demand into the routing decision by comparing each router trained on the joint objective (integrated) with the same router trained on cost alone and without demand information (separate). Integration improved every metric for both routers (Holm-corrected $p < 0.001$). For the neural router, cost per 1000 units fell from 0.339 to 0.227 (33% lower), demand satisfaction rose from 0.498 to 0.554, and mean delay fell from 0.333 h to 0.024 h; for the tabular router, cost per unit fell by 45%. Integration is therefore the largest effect of any architectural component (the neural router without demand information was 12% cheaper per unit than reactive IoT-only dispatch, and validation leaves valid plans unchanged), and it is of the same order as replacing the decision engine, whose effect ranges from 27% (tabular to neural) to 37% (tabular to ALNS).

Table 11. Impact of integrating demand into the routing decision (mean ± SD; Holm-adjusted Wilcoxon p, 30 paired seeds).

Router	Objective	Total Cost	Cost per 1000 Units	Demand Satisfaction	Delivery Delay (h)
Tabular	Separate (cost only)	24.10 ± 2.14	0.568 ± 0.146	0.377 ± 0.070	0.713 ± 0.177
Tabular	Integrated (joint)	19.70 ± 1.39	0.311 ± 0.027	0.535 ± 0.009	0.243 ± 0.100
Tabular	Holm-adjusted p	$p < 0.001$	$p < 0.001$	$p < 0.001$	$p < 0.001$
Neural	Separate (cost only)	19.79 ± 1.89	0.339 ± 0.047	0.498 ± 0.027	0.333 ± 0.114
Neural	Integrated (joint)	14.96 ± 0.99	0.227 ± 0.015	0.554 ± 0.005	0.024 ± 0.008
Neural	Holm-adjusted p	$p < 0.001$	$p < 0.001$	$p < 0.001$	$p < 0.001$

The cost-only objective underperforms even on total cost. It steers the router toward short hops to small customers (19.0 versus 8.4 stops per day for the neural router), which fills the vehicle slowly and exhausts the shift before capacity is used. Rewarding delivered quantity at every stop lets the router trade distance against load directly, which is the mechanism that the joint formulation (7) encodes. Figure 8 shows the comparison graphically.

4.4. Learning Behavior and Training Diagnostics

Figure 9 shows cost per 1000 units and demand satisfaction during training, which includes exploration, and Figure 10 shows diagnostics that separate learning from exploration. The greedy policy evaluated every 80 days on 20 validation days (Figure 10a) shows that the neural router fell below the demand-aware heuristic's level from the first checkpoint and settled at 0.228 by the end of training; its last three checkpoints varied by 1.3% within a seed, and the improvement after day 400 was 1.1%. The tabular router plateaued near 0.312, above the heuristic, which is consistent with its state being unable

to tell which customers remain. The Huber temporal-difference loss of the neural router decreased steadily over 8405 gradient updates, from a peak of 0.063 to 0.029 (Figure 10b), while the mean predicted Q-value rose to about 1.0 without diverging, and exploration followed the linear schedule to 0.05 by day 480 (Figure 10c).

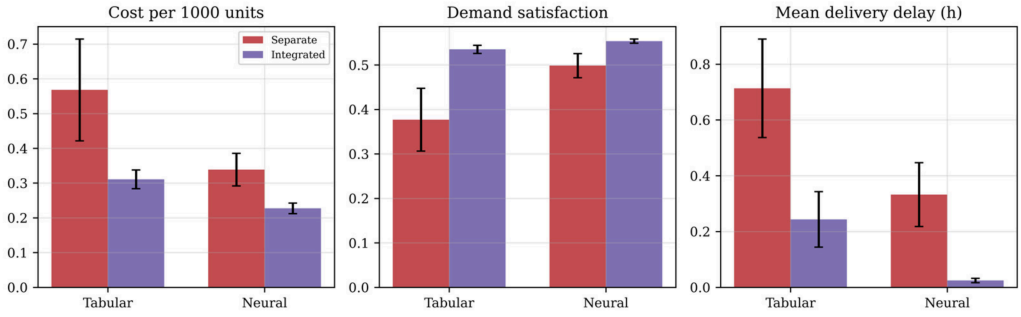


Figure 8. Separate versus integrated reward for the tabular and neural routers (mean ± SD over 30 seeds).

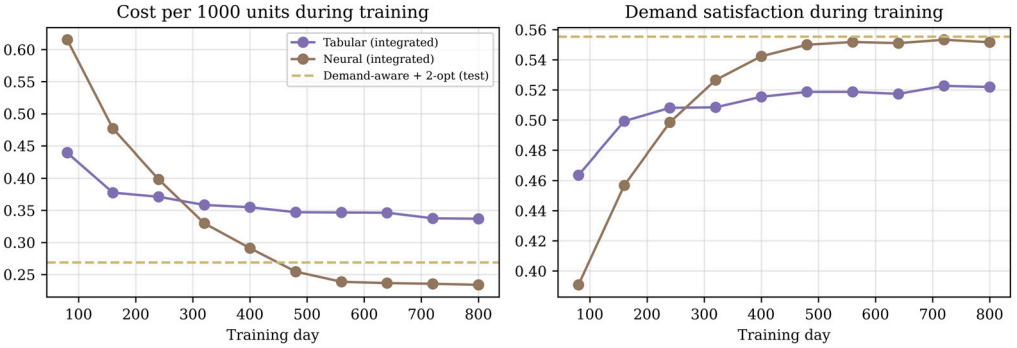


Figure 9. Training curves: cost per 1000 units and demand satisfaction (30 seeds).

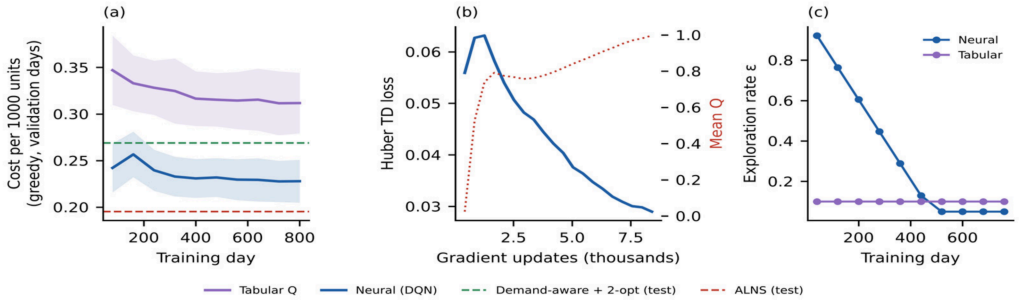


Figure 10. (**a**) greedy validation cost, mean ± SD (shaded); (**b**) TD loss and mean Q; (**c**) exploration rate (30 seeds).

Because the neural engine's settings were common defaults, Table S1 (Supplementary Materials) varies each of them in turn. Across the twelve alternative settings, its cost per 1000 units stayed within −1.1% to +3.7% of the default, and it remained 12.0% to 16.0% below the demand-aware heuristic in every seed for every setting. The advantage therefore does not depend on a tuned configuration. With the discount factor $\gamma = 0.99$, for which the return is closest to the undiscounted objective, cost per unit was 0.233 (12.0% below the heuristic, 10 of 10 seeds), 3.7% above the default $\gamma = 0.9$.

4.5. Additional Routing Constraints

The baseline model treats due times as soft. Table S2 (Supplementary Materials) reports the two standard extensions that the formulation allows: hard time windows (16),

enforced in the admissible set of the learned routers, in the ALNS feasibility check and by the validators, and, in addition, a heterogeneous fleet with trip capacities of 0.5, 1.0 and 1.5 times the mean. Every engine produced only feasible plans (no plan was rejected and no delivery was late), and the ranking of Section 4.1 was unchanged: with hard windows the learned engine with rollout was 25.2% cheaper per unit than the demand-aware heuristic, and ALNS seeded by it was 28.1% cheaper and below ALNS alone in 10 of 10 seeds; with the heterogeneous fleet the figures were 24.9% and 28.0% (10 of 10 seeds in every comparison with the heuristic). Without rollout, the learned router was 16.1% and 16.5% cheaper than the heuristic. Because the windows remove late stops, every engine became slightly cheaper per unit than in the soft-window setting, and post-2-opt of the learned trips is now constrained by the windows.

4.6. Coverage–Cost Trade-Off

The demand-aware engines serve fewer customers per day than nearest-neighbor dispatch (8.4 and 7.8 against 24.5), and over 200 days the neural engine served 88% of the customers at least once (Table 9). An operator with service commitments to small customers can move along this trade-off through the coverage weight α_4 of (7), which rewards every customer served. Figure 11 traces the resulting cost–coverage frontier for the neural engine and ALNS (α_4 from 0 to 1) and for a coverage-weighted version of the demand-aware heuristic, evaluated on the first 60 held-out days of 10 seeds (ALNS with 50 iterations per day).

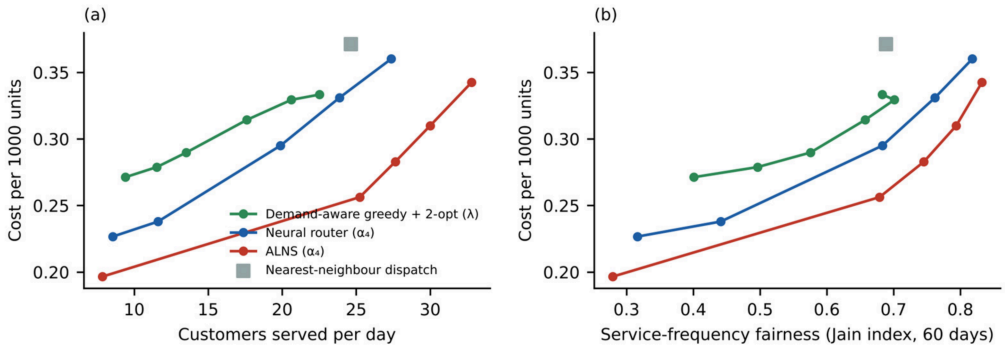


Figure 11. Cost–coverage frontier: (**a**) customers per day; (**b**) service fairness (10 seeds, 60 days).

Raising α_4 from 0 to 1 increased the customers served per day from 8.6 to 27.4 for the neural engine and from 7.8 to 32.8 for ALNS, and the share of customers served at least once from 79% to 98% and from 70% to 100%. At the daily coverage of nearest-neighbor dispatch (24.6 customers per day, 0.371 per 1000 units), ALNS cost 0.254 per 1000 units, 31% less, and the neural engine 0.337, 9% less; at that coverage ALNS also served more customers over the horizon (96% against 91%). The decision-centric engines therefore deliver lower cost per unit at matched coverage: at every level of coverage that dispatch or the coverage-weighted heuristic reaches, the ALNS frontier lies at lower cost per unit. Coverage is a policy lever with a measurable price, which the operator sets through α_4, or through a minimum-service rule added to the shared constraints, according to its service commitments.

4.7. Robustness to the Simulation Assumptions

To assess robustness, the policies trained at the baseline noise level were evaluated (12 seeds, 100 days per level) with IoT demand-observation noise from 0% to 40% and a missing-reading probability rising from 0% to 10%. As noise increased, demand satisfaction fell for every configuration (Spearman ρ between -0.47 and -0.89, $p < 0.001$), and cost per

1000 units rose for the learned routers (ρ = 0.30 tabular and 0.36 neural; $p < 0.05$) while changing little for the heuristics. The proposed system's advantage over the demand-aware heuristic narrowed gradually, from 16.6% at 0% noise to 14.0% at 40%, and remained significant at every level ($p < 0.001$; lower cost per unit in 12 of 12 seeds).

Figure 12 shows both metrics across noise levels. The policies were not retrained at each level, so the results measure robustness to a shift in observation quality after deployment.

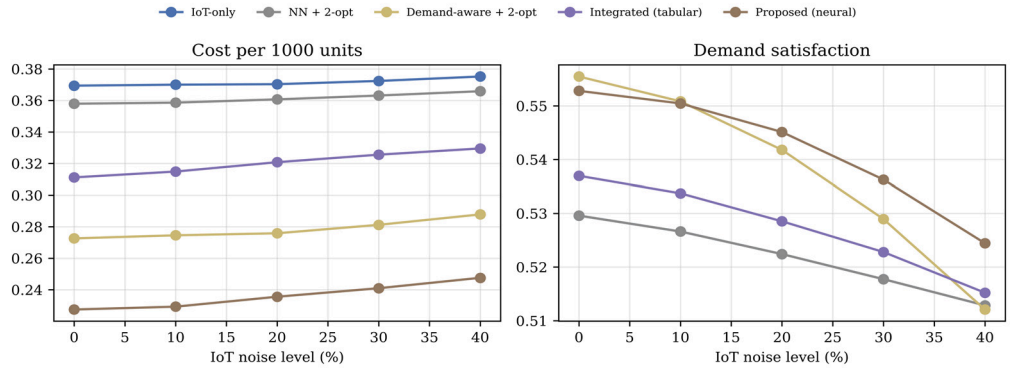


Figure 12. Sensitivity of cost per 1000 units and demand satisfaction to IoT noise (12 seeds).

The network layout, fleet, capacity rule, due times and local cost rate are assumptions, because the dataset has no customer locations. Table 12 changes each of them in turn and retrains the learned router in every scenario. The learned engine with rollout was cheaper per unit than the demand-aware heuristic in all 12 settings (the baseline and 11 alternatives) and in all 120 paired runs, by 11.5% to 31.7% (Holm-adjusted $p < 0.05$ in every setting), with its largest margins when test demand rose 30% above the training level without retraining (31.7%), when capacity was scarce (31.5%) and when customers were clustered (30.7%). Even at four times the baseline cost per km, where the router without rollout had no advantage (+1.8%, 5 of 10 seeds), it was 11.5% cheaper in 10 of 10 seeds. ALNS seeded by the learned plan was 16.0% to 34.3% cheaper than the heuristic and below ALNS alone in every setting (112 of 120 runs), giving the lowest cost per unit in all 12 settings. When capacity covered all mean demand, rollout traded demand satisfaction for cost (0.793 against 0.848), whereas the seeded engine served the most demand (0.862) at 32.8% lower cost per unit. The advantage of the proposed engines therefore does not depend on the assumed parameters.

Table 12. Robustness to the assumed operating parameters (one factor at a time; 10 seeds, 100 held-out days; routers retrained in each scenario).

Scenario	Demand-Aware + 2-opt	Neural	Neural + Roll-out	ALNS	Seeded ALNS	Rollout vs. Heuristic, % (Seeds Better)	Seeded ALNS vs. Heuristic, % (Seeds Better)	Seeded ALNS vs. ALNS, % (Seeds Better)	Demand Satisfaction: Heuristic/Rollout/Seeded ALNS
Baseline (as Table 5)	0.265	0.225	0.202	0.195	0.193	+23.6 (10/10)	+27.2 (10/10)	+1.1 (10/10)	0.555/0.555/0.555
Capacity 0.40 × mean demand	0.308	0.234	0.211	0.211	0.207	+31.5 (10/10)	+32.8 (10/10)	+1.7 (10/10)	0.404/0.404/0.404
Capacity 0.70 × mean demand	0.273	0.245	0.216	0.197	0.195	+20.7 (10/10)	+28.3 (10/10)	+0.6 (9/10)	0.695/0.687/0.700
Capacity 1.00 × mean demand	0.292	0.261	0.237	0.199	0.196	+18.7 (10/10)	+32.8 (10/10)	+1.4 (9/10)	0.848/0.793/0.862

Table 12. *Cont.*

Scenario	Demand-Aware + 2-opt	Neural	Neural + Rollout	ALNS	Seeded ALNS	Rollout vs. Heuristic, % (Seeds Better)	Seeded ALNS vs. Heuristic, % (Seeds Better)	Seeded ALNS vs. ALNS, % (Seeds Better)	Demand Satisfaction: Heuristic/Rollout/Seeded ALNS
2 trips per day	0.220	0.194	0.172	0.163	0.162	+21.8 (10/10)	+26.4 (10/10)	+0.7 (8/10)	0.555/0.555/0.555
5 trips per day	0.356	0.296	0.260	0.254	0.249	+26.9 (10/10)	+29.9 (10/10)	+1.7 (9/10)	0.551/0.551/0.550
Clustered customer layout	0.229	0.177	0.159	0.155	0.153	+30.7 (10/10)	+33.4 (10/10)	+1.8 (10/10)	0.556/0.555/0.555
Region 200 × 200 km	0.356	0.329	0.305	0.292	0.288	+14.4 (10/10)	+19.2 (10/10)	+1.5 (9/10)	0.543/0.547/0.550
Local cost per km × 2	0.357	0.332	0.295	0.289	0.284	+17.3 (10/10)	+20.5 (10/10)	+1.6 (10/10)	0.555/0.555/0.555
Local cost per km × 4	0.542	0.532	0.480	0.460	0.455	+11.5 (10/10)	+16.0 (10/10)	+1.0 (9/10)	0.555/0.555/0.552
Due times × 0.67	0.265	0.227	0.204	0.197	0.194	+22.8 (10/10)	+26.5 (10/10)	+1.3 (9/10)	0.555/0.554/0.555
Test demand +30% (no retraining)	0.255	0.197	0.174	0.169	0.167	+31.7 (10/10)	+34.3 (10/10)	+1.1 (10/10)	0.428/0.428/0.428

Columns 2–6: cost per 1000 units delivered (mean). Rollout, K = 4; seeded ALNS, ALNS started from the rollout plan.

4.8. Scalability and the Quality–Latency Trade-Off

To test whether the neural router's fixed-size representation matters in practice, both learned routers and the demand-aware heuristic were evaluated on networks of 10 to 160 customers bootstrapped from the real customer pool, with the fleet scaled proportionally (5 seeds per size, 400 training and 100 test days). Table 13 and Figure 13 show that the tabular Q-table grew from 199 to 6152 entries, about 31-fold for a 16-fold increase in network size, while the neural router kept 1441 parameters. On the smallest networks (10 and 20 customers, one vehicle), the neural router had no consistent advantage over the demand-aware heuristic, with lower cost per unit in 1 of 5 and 2 of 5 seeds. From 40 customers upward, it was better in every seed, and its advantage grew with size, from 14.9% at 40 customers to 20.3% at 160. The tabular router was worse than the heuristic at every size.

Table 13. Scalability (customers bootstrapped from the real pool; 5 seeds per size; mean).

Customers	Vehicles	Q-Table Entries	Neural Parameters	Cost/1000 Units: Tabular	Neural	Demand-Aware + 2-opt	Neural vs. Heuristic
10	1	199	1441	0.919	0.696	0.731	+4.8% (1/5)
20	1	513	1441	0.354	0.306	0.304	−0.6% (2/5)
40	3	1379	1441	0.324	0.235	0.276	+14.9% (5/5)
80	6	2893	1441	0.318	0.201	0.241	+16.7% (5/5)
160	12	6152	1441	0.332	0.206	0.258	+20.3% (5/5)

Figure 14 places the engines on the quality–latency plane: ALNS with wall-clock budgets from 0.02 to 2 s per day, and the learned engine without rollout, with 2-opt and with rollout over K = 2, 4 and 8 candidates (5 seeds, 30 days per network size; the networks of Table 13). Comparing each ALNS budget with the best learned variant whose mean decision time fits within it, the learned engine gave the lower cost per unit at every budget from 0.05 s to 0.1 s with 41 customers, from 0.02 s to 1 s with 80 customers and from 0.05 s to 2 s with 160 customers (a near tie at 0.1 s on 41 customers; a near tie at 1 s on 160 customers). At the shortest budgets, the winning variant is the learned router itself, which decides in 2 to 25 ms; at longer budgets it is rollout: on 80 customers, rollout with K = 8 reached 0.175 per 1000 units in 400 ms against 0.177 for ALNS with 1 s, and on 160 customers,

rollout with K = 4 reached 0.189 in 1.9 s against 0.199 for ALNS with 2 s. ALNS did better only on the smaller networks: on 41 customers at 0.02 s and from 0.2 s upward, and on 80 customers at 2 s; there, seeding it with the learned plan improved it further (Section 4.1). The learned engine's advantage therefore grows with network size: within a fixed time, search must explore a combinatorially larger neighborhood, whereas the learned policy directs the rollout to the few candidates worth evaluating. The decision layer follows the anytime pattern of intelligent systems [39]: the learned engine supplies a strong decision within milliseconds, and additional time is converted into quality by rollout or by search seeded with the learned plan, all behind the same state, objective and validation interfaces.

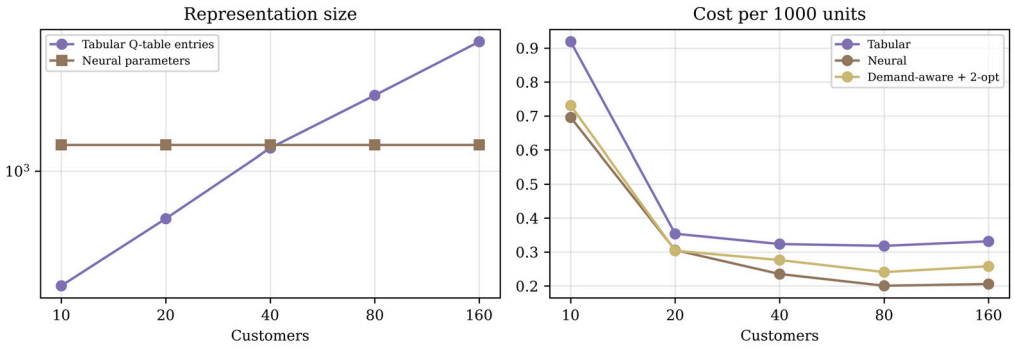


Figure 13. Representation size and cost per 1000 units versus network size (5 seeds per size).

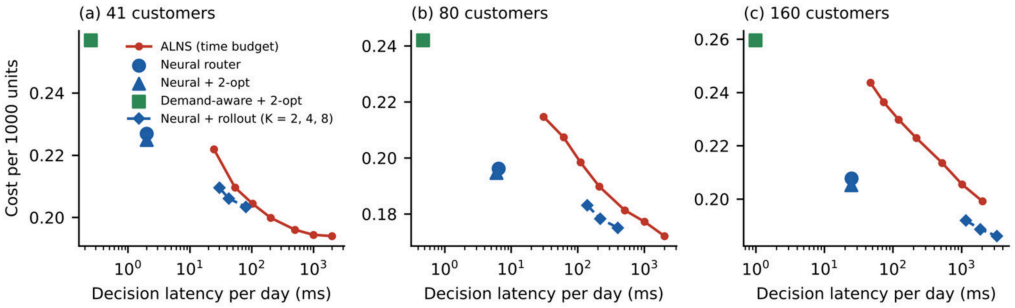


Figure 14. Decision quality versus decision latency for 41, 80 and 160 customers.

4.9. Blockchain Validation Cost and Scalability

Table 14 reports the measured cost of validation in the evaluated five-validator ledger. These are processing times only: the validators run in one process and exchange no messages. Validation, consensus, trust update and hashing took 0.15 to 0.30 ms per daily plan, 0.15 ms (5.9%) of the neural engine's decision latency and a negligible share of the ALNS engine's. No valid plan was rejected in any configuration, and every chain verified as intact. Figure 15 splits the ledger time into its validation and consensus components.

The evaluated ledger runs its validators in one process and therefore omits communication. Table S3 and Figure S1 (Supplementary Materials) estimate that cost by discrete-event simulation of PBFT (Section 3.3.4): the per-message processing costs are measured on the evaluation machine (52 µs per Ed25519 signature, 151 µs per verification, 23 µs per constraint check), while network delays are assumed to follow log-normal distributions. In this simulation, a four-validator network committed a daily plan in 1.4 ms on a local network and 62 ms over a regional wide-area network, about three one-way delays, as the protocol requires. Message count grows quadratically, from 24 to 19,800 per decision between 4 and 100 validators, and per-node processing grows linearly, to 30 ms at 100 validators, so on a local network latency rose to 26 ms. Over wide-area networks, latency is dominated by propagation: 73 ms regionally and 280 ms globally with 100 validators,

and 135 ms regionally with 33 of them silent. Even the largest simulated configuration committed a plan in well under a second, several orders of magnitude inside a daily or hourly planning cycle. Under the modeled delays, consensus latency would not constrain the decision cadence of a supply chain consortium, and per-node signature verification, not consensus, is the binding resource at scale; confirming this on an operational permissioned network is part of the next step (Section 5.4).

Table 14. Measured blockchain validation cost per daily plan (wall-clock, mean over 30 seeds).

Configuration	Validation (ms)	Consensus and Hashing (ms)	Ledger Total (ms)	Decision Latency (ms)	Ledger Share	Chains Intact	Valid Plans Rejected
Blockchain-only	0.139	0.083	0.222	0.81	27.6%	30/30	0
Separate, tabular	0.108	0.073	0.180	1.00	18.1%	30/30	0
Proposed, tabular	0.083	0.066	0.149	0.68	21.9%	30/30	0
Separate, neural	0.128	0.091	0.220	2.83	7.8%	30/30	0
Proposed, neural	0.075	0.076	0.151	2.55	5.9%	30/30	0
Proposed, ALNS	0.070	0.232	0.302	917	<0.1%	30/30	0

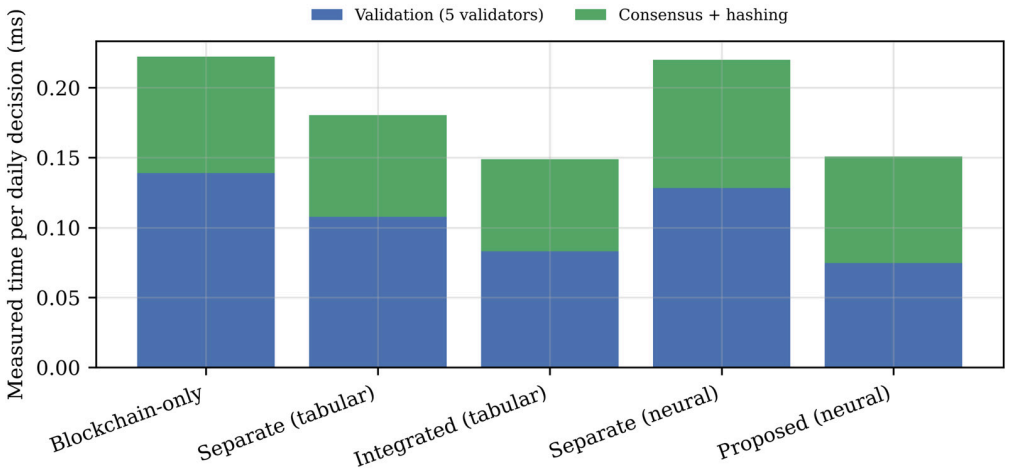


Figure 15. Measured ledger time per daily decision, split into validation and consensus.

4.10. Ledger Security and Containment of Decision Faults

Three experiments test whether the ledger governs decisions rather than only recording them. In the tamper experiment, 200 randomly chosen blocks per seed of a 200-day chain were altered, over 30 seeds. An edit without re-hashing and an edit with the altered block re-hashed were both detected in 100% of trials, because the next block's back-link no longer matches. An attacker who re-hashes the entire suffix of a single copy is not detected by that copy (0%). Detection then depends on comparing replicas, and a consistent rewrite of a majority of the five validators' copies would be accepted.

The injection experiment corrupted 10% of daily plans and made one to four of the five validators faulty, either approving every plan or rejecting every plan (Table 15). Under the majority rule of the initial submission, invalid plans were accepted in 100% of cases once approving validators held a majority, valid plans were rejected in 100% of cases once rejecting validators did, and the agreement-based trust score did not isolate validators that approved everything. Under the evidence rule (38), no invalid plan was accepted (0.0%), and no valid plan was rejected (0.0%) for any number of faulty validators up to four of five and for both behaviors, because one honest validator's verified witness rejects an invalid plan and a reject vote without a verifiable witness carries no weight. The verified-outcome trust also separated the validators: the trust of validators that rejected everything fell to 0.10, and they were excluded, while validators that approved everything kept a trust

of 0.90, because they are wrong only on the corrupted days, but could no longer cause an invalid plan to be accepted. Safety thus requires one honest validator instead of an honest majority.

Table 15. Majority rule versus evidence rule under faulty validators (10% corrupted plans, 500 days, 30 seeds).

Faulty Behavior	Faulty Validators	Majority: Invalid Accepted	Majority: Valid Rejected	Evidence: Invalid Accepted	Evidence: Valid Rejected	Evidence: Final Trust, Faulty	Evidence: Final Trust, Honest
Approve all	1 of 5	0.0%	0.0%	0.0%	0.0%	0.92	1.00
	2 of 5	0.0%	0.0%	0.0%	0.0%	0.90	1.00
	3 of 5	100.0%	0.0%	0.0%	0.0%	0.88	1.00
	4 of 5	100.0%	0.0%	0.0%	0.0%	0.90	1.00
Reject all	1 of 5	0.0%	0.0%	0.0%	0.0%	0.08	1.00
	2 of 5	0.0%	0.0%	0.0%	0.0%	0.10	1.00
	3 of 5	0.0%	100.0%	0.0%	0.0%	0.12	1.00
	4 of 5	0.0%	100.0%	0.0%	0.0%	0.10	1.00

The third experiment corrupted 1% to 20% of the neural engine's daily plans with one of four constraint violations (capacity overload, duplicate visit, shift overrun or non-existent customer). Without validation every corrupted plan was executed; with validation none was, at any fault rate. Replacing a rejected plan by reactive dispatch raised cost per unit by 7.3% at a 10% fault rate, whereas validate-and-repair (39), which removes only the offending stops, kept the increase to 0.4%, so engine faults become a small, bounded loss of efficiency instead of executed constraint violations.

5. Discussion

5.1. Key Insights

The evaluation supports four conclusions for the design of decision-centric supply chain systems. First, constraint safety is a property of the loop rather than of the decision engine: validation executed no invalid plan under engine faults of up to 20%, needed only one honest validator under the evidence rule, and with validate-and-repair cost 0.4% of cost per unit at a 10% fault rate. Second, what the system optimizes is its strongest lever: integrating delivered demand into the objective lowered cost per unit by 33–45%, the largest effect of any architectural component (the learned engine without demand information lowered it by 12% relative to reactive dispatch, and validation leaves valid plans unchanged). Third, the learned engine is the fast core of the decision layer: on its own at short budgets and with decision-time rollout at longer ones, it achieved lower cost than ALNS at every search budget from 0.05 s to 0.1 s on 41 customers, 0.02 s to 1 s on 80 customers and 0.05 s to 2 s on 160 customers (mean decision time), and seeding search with its plan gave the lowest cost overall and in every one of the 13 alternative operating settings. Fourth, the learned engine's representation determined whether learning paid off: the tabular router, whose state cannot tell which customers remain, plateaued above the heuristic, whereas the candidate-scoring network kept 1441 parameters from 10 to 160 customers and its advantage grew with network size.

5.2. Managerial and Industrial Implications

For supply chain managers the results translate into four decisions. The first is the performance measure. When capacity is scarce, routing on distance or cost alone serves the wrong customers: the cost-only router delivered 49.8% of demand at 0.339 per 1000 units, against 55.4% at 0.227 under the joint objective. Dispatch rules and planning KPIs should therefore price delivered demand and lateness together with cost. The second is the

decision engine. The learned engine, with rollout at longer budgets, gave lower cost than search at every budget from 0.05 s to 0.1 s on 41 customers, 0.02 s to 1 s on 80 customers and 0.05 s to 2 s on 160 customers (mean decision time), and with rollout it was 24.3% cheaper per unit than the demand-aware rule in 43 ms on the evaluated network; when seconds are available, as in daily planning, seeding search with the learned plan was 28.0% cheaper than the demand-aware rule and 31.5% cheaper than a general-purpose solver. The third is the service policy. Serving fewer, larger customers lowers cost per unit, and the coverage frontier of Section 4.6 tells the operator what each additional customer served per day costs, so service commitments can be priced rather than discovered. The fourth is governance. The validation layer adds about 0.15 ms per decision, works with existing dispatch rules (the Blockchain-only configuration left every outcome unchanged), and turns the operating terms agreed among partners into checks that every plan must pass.

Adoption can proceed in stages that limit risk. (1) Codify the shared constraints (capacities, shift rules, delivery windows, valid customers) with carriers and customers and run the validation layer alongside the current dispatch process; this gives an auditable record of every plan at negligible cost. (2) Integrate the IoT and order data that the state requires and run the decision engine in shadow mode, comparing its plans with the dispatcher's on the operator's own network; the network-size results suggest that a learned engine pays off from about 40 customers, whereas search paid off at every size at which it was run (41 to 160 customers). (3) Release validated plans automatically with validate-and-repair as the fallback, and keep a human dispatcher for rejected or repaired plans. (4) Extend validation to a consortium of partners with a Byzantine fault-tolerant protocol, which the PBFT simulation of Section 4.9 indicates is fast enough for daily or hourly planning.

The main challenges are organizational rather than computational. Partners must agree on the constraint set and on who operates validators, which requires governance of a shared system among firms with different incentives. Sensor coverage and data quality determine the state the engines see; the noise study shows graceful degradation, but missing data on whole customer segments would bias decisions. The decision engines must be integrated with existing transport and enterprise systems, and the learned engine must be retrained when the network changes substantially, although it tolerated a 30% demand increase without retraining. Finally, dispatchers and partners need to trust automated decisions; the validation record, which shows why each plan was accepted, repaired or rejected, supports this, and explanations of the engines' choices are a natural extension.

5.3. Sustainability

Cost and distance must be priced together. Measured in vehicle-km per 1000 units delivered, a proxy for local transport emissions for a given vehicle type, the demand-aware heuristic needed 4.77 km, ALNS 5.07 km the neural engine 5.96 km and the neural engine with rollout 5.24 km (Tables 8 and 9); the learned engine reached its lower cost by consolidating freight, which a distance price steers toward shorter routes. The framework can make emissions an explicit objective in three ways. A carbon price can be added to the per-km cost: in the robustness runs of Section 4.7, at four times the baseline per-km cost, ALNS reduced its distance per unit from 4.98 to 4.37 km, below the heuristic's 4.63 km, while remaining 15.1% cheaper per unit, and the neural engine's distance fell from 5.88 to 5.16 km. An emissions term weighted by vehicle type and load can be added to the objective (7) as a further weighted criterion. An emissions cap can be added to the shared constraint set, so that the validation layer rejects or repairs plans that exceed an agreed budget and the ledger records the emissions of every executed plan, which is the accounting that carbon-

trading strategies require; recent work identifies the decarbonization of transportation as central to such strategies [40].

5.4. Limitations and Future Work

Several limitations remain. The IoT streams and the validator network are simulated: the ledger timings of Table 14 exclude communication, and the PBFT study of Section 4.9 models it with measured processing costs and assumed network delays rather than a deployment on a production platform. The dataset is a single-period snapshot whose bootstrap-resampled demand stands in for a multi-period series and has no customer locations, so the layout and operating parameters are assumptions; Section 4.7 shows which conclusions depend on them. The study covers one depot, daily planning and a single product flow, and the emissions measure is a distance proxy. Validating the system on vehicle and sensor hardware, on an operational permissioned network and on data with genuine temporal and geographic structure is the most important next step.

Three technical extensions follow directly from the results. Rollout and seeding improve the learned engine at decision time without retraining; training the policy on the improved decisions (approximate policy iteration, or imitation of the seeded search) could transfer part of that gain to the millisecond engine, and comparison with attention-based learned routers [18,20] on the same instances would place it among learned methods. The ledger detects tampering with a single copy but relies on replica comparison or external anchoring against a consistent rewrite of a majority of replicas. Extending the framework to multi-tier, multi-depot supply chains with partners in different regulatory regimes, and adding explanations of the engines' decisions, would broaden its practical use.

6. Conclusions

This paper treated the supply chain information system as a closed decision loop in which IoT data acquisition, an AI decision engine, and blockchain-based validation are designed and evaluated together. The daily routing problem was formulated as a prize-collecting capacitated vehicle routing problem whose objective equals the undiscounted return of the sequential process solved by the learned routers, so learned and search engines are interchangeable within the loop, and every plan is validated against the partners' shared constraints, recorded, and repaired or rejected before execution.

In a simulation grounded in a real multi-plant distribution dataset, over 30 seeds with paired tests, the learned deep Q-network engine with decision-time rollout reduced cost per 1000 units delivered by 24.3% relative to a demand-aware heuristic and 28.0% relative to OR-Tools while deciding in 43 ms, and it achieved lower cost than ALNS at every search budget from 0.05 s to 0.1 s on 41 customers, 0.02 s to 1 s on 80 customers and 0.05 s to 2 s on 160 customers (mean decision time), on its own at short budgets and with rollout at longer ones. Seeding ALNS with the learned plan gave the lowest cost overall, 28.0% below the heuristic and below ALNS alone in 29 of 30 seeds. Integrating demand into the routing objective lowered cost per unit by 33–45% relative to a cost-only objective, the largest effect of any architectural component (the learned engine without demand information lowered it by 12% relative to reactive dispatch, and validation leaves valid plans unchanged). Under hard time windows, a heterogeneous fleet and eleven alternative operating scenarios, the learned engine with rollout was 12–32% cheaper than the heuristic in all 130 paired runs, and seeding ALNS with its plan gave the lowest cost in every setting. Evidence-based validation kept every invalid plan from execution with up to four of five validators faulty and under engine faults of up to 20%, at 0.4% of cost per unit at a 10% fault rate, and, in a PBFT simulation with measured processing costs, Byzantine fault-tolerant validation with 100 validators committed a plan in well under

a second. For practitioners, the results indicate that the routing objective should price delivered demand, that a learned engine should make time-critical decisions, using rollout when tens of milliseconds are available and seeding search when seconds are, and that the validation layer can be adopted first, alongside existing dispatch, at negligible cost. The released code regenerates every reported number.

Supplementary Materials: Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/systems14111393/s1, Table S1: Sensitivity of the neural engine to its hyperparameters and reward weights (one factor at a time; 10 seeds, 100 held-out days); Table S2: Routing with hard time windows and a heterogeneous fleet (10 seeds, 100 held-out days, validation on); Table S3: Byzantine fault-tolerant validation at deployment scale (PBFT simulation with measured processing costs and assumed network delays; median of 100 decisions); Figure S1: PBFT validation versus validator count: (a) commit latency (dashed: faulty validators silent); (b) messages and processing per node; Software S1: simulation and analysis code.

Author Contributions: Conceptualization, T.H. and F.H.M.; methodology, T.H. and F.H.M.; software, T.H. and F.H.M.; validation, T.H. and F.H.M.; formal analysis, T.H. and F.H.M.; investigation, T.H. and F.H.M.; data curation, T.H. and F.H.M.; writing—original draft preparation, T.H. and F.H.M.; writing—review and editing, T.H. and F.H.M.; visualization, T.H. and F.H.M. All authors have read and agreed to the published version of the manuscript.

Funding: The authors gratefully acknowledge the funding of the Deanship of Graduate Studies and Scientific Research, Jazan University, Saudi Arabia, through Project number: RP2026-090.

Data Availability Statement: The data presented in this study are available in the Brunel University London research data repository at https://doi.org/10.17633/rd.brunel.7558679.v2 [36]. These data were derived from the public-domain supply chain logistics problem dataset. **Code Availability Statement:** The complete reproducibility recipe and accompanying code are publicly archived at https://github.com/thakim-code/SCIS (accessed on 15 September 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	artificial intelligence
ALNS	adaptive large neighborhood search
BFT	Byzantine fault tolerance
CRF	customer-referred freight
DQN	deep Q-network
DTD	door-to-door
DTP	door-to-port
EWMA	exponentially weighted moving average
GLS	guided local search
GPS	global positioning system
IoT	Internet of Things
KPU	vehicle-km per 1000 units delivered
LAN	local area network
MLP	multilayer perceptron
NN	nearest neighbor
PBFT	Practical Byzantine Fault Tolerance
ReLU	rectified linear unit
RFID	radio-frequency identification
RL	reinforcement learning
SCIS	supply chain information system(s)
SD	standard deviation

TD	temporal difference
UC	unit cost (cost per 1000 units delivered)
WAN	wide area network

References

1. Zrelli, I.; Rejeb, A. A bibliometric analysis of IoT applications in logistics and supply chain management. *Heliyon* **2024**, *10*, e36578. [CrossRef] [PubMed]
2. Tavana, M.; Khalili Nasr, A.; Ahmadabadi, A.B.; Amiri, A.S.; Mina, H. An interval multi-criteria decision-making model for evaluating blockchain-IoT technology in supply chain networks. *Internet Things* **2023**, *22*, 100786. [CrossRef]
3. Yarlagadda, K.C. AI-powered supply chain optimization: Enhancing demand forecasting and logistics. *J. Comput. Sci. Technol. Stud.* **2025**, *7*, 792–801. [CrossRef]
4. Khan, P.W.; Byun, Y.C.; Park, N. IoT-blockchain enabled optimized provenance system for food industry 4.0 using advanced deep learning. *Sensors* **2020**, *20*, 2990. [CrossRef] [PubMed]
5. Ben-Daya, M.; Hassini, E.; Bahroun, Z. Internet of things and supply chain management: A literature review. *Int. J. Prod. Res.* **2019**, *57*, 4719–4742. [CrossRef]
6. Arvind, V.R.; Shrinidhi, R.M.; Deepa, T.; Maheedhar, M. Intelligent warehousing: A machine learning and IoT framework for precision inventory optimization. *IEEE Access* **2025**, *13*, 169381–169414. [CrossRef]
7. Bhupathi, H.; Chinta, S.; Bhagyalakshmi, L.; Kishorekumar, M.; Devarasu, N.; Venu, K. The potential of embedded systems in optimizing industrial logistics. In Proceedings of the 2024 IEEE 4th International Conference on ICT in Business Industry & Government (ICTBIG), Indore, India, 13–14 December 2024. [CrossRef]
8. Humayun, M.; Jhanjhi, N.Z.; Hamid, B.; Ahmed, G. Emerging smart logistics and transportation using IoT and blockchain. *IEEE Internet Things Mag.* **2020**, *3*, 58–62. [CrossRef]
9. Khan, D.; Jung, L.T.; Hashmani, M.A.; Cheong, M.K. Empirical performance analysis of Hyperledger LTS for small and medium enterprises. *Sensors* **2022**, *22*, 915. [CrossRef] [PubMed]
10. Mangala, N.; Naveen, D.R.; Reddy, B.E.; Buyya, R.; Venugopal, K.R.; Iyengar, S.S.; Patnaik, L.M. Secure pharmaceutical supply chain using blockchain in IoT cloud systems. *Internet Things* **2024**, *26*, 101215. [CrossRef]
11. Nanda, S.K.; Panda, S.K.; Dash, M. Medical supply chain integrated with blockchain and IoT to track the logistics of medical products. *Multimed. Tools Appl.* **2023**, *82*, 32917–32939. [CrossRef] [PubMed]
12. Azizi, N.; Malekzadeh, H.; Akhavan, P.; Haass, O.; Saremi, S.; Mirjalili, S. IoT–blockchain: Harnessing the power of internet of thing and blockchain for smart supply chain. *Sensors* **2021**, *21*, 6048. [CrossRef] [PubMed]
13. Ugochukwu, N.A.; Goyal, S.B.; Arumugam, S. Blockchain-based IoT-enabled system for secure and efficient logistics management in the era of IR 4.0. *J. Nanomater.* **2022**, *2022*, 7295395. [CrossRef]
14. Brandín, R.; Abrishami, S. IoT-BIM and blockchain integration for enhanced data traceability in offsite manufacturing. *Autom. Constr.* **2024**, *159*, 105266. [CrossRef]
15. Chen, J.; Xu, S.; Liu, K.; Yao, S.; Luo, X.; Wu, H. Intelligent transportation logistics optimal warehouse location method based on internet of things and blockchain technology. *Sensors* **2022**, *22*, 1544. [CrossRef] [PubMed]
16. Rizwan, A.; Karras, D.A.; Kumar, J.; Sánchez-Chero, M.; Mogollón Taboada, M.M.; Altamirano, G.C. An internet of things (IoT) based block chain technology to enhance the quality of supply chain management (SCM). *Math. Probl. Eng.* **2022**, *2022*, 9679050. [CrossRef]
17. Alharby, M. Preserving data secrecy and integrity for cloud storage using smart contracts and cryptographic primitives. *Comput. Mater. Contin.* **2024**, *79*, 2449–2463. [CrossRef]
18. Pan, W.; Liu, S.Q. Deep reinforcement learning for the dynamic and uncertain vehicle routing problem. *Appl. Intell.* **2023**, *53*, 405–422. [CrossRef]
19. Zheng, G.; Kong, L.; Brintrup, A. Federated machine learning for privacy preserving, collective supply chain risk prediction. *Int. J. Prod. Res.* **2023**, *61*, 8115–8132. [CrossRef]
20. Kool, W.; van Hoof, H.; Welling, M. Attention, learn to solve routing problems! In Proceedings of the 7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, 6–9 May 2019. Available online: https://openreview.net/forum?id=ByxBFsRqYm (accessed on 20 September 2026).
21. Raouf, Y.; Benmamoun, Z.; Hachimi, H. An artificial intelligence framework for recycling dormant and obsolete inventory in supply chains. *Supply Chain Anal.* **2025**, *11*, 100152. [CrossRef]
22. Ropke, S.; Pisinger, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transp. Sci.* **2006**, *40*, 455–472. [CrossRef]
23. Perron, L.; Furnon, V. *OR-Tools*, version 9.15; Google: Mountain View, CA, USA. Available online: https://developers.google.com/optimization (accessed on 16 October 2026).

24. Toth, P.; Vigo, D. (Eds.) *Vehicle Routing: Problems, Methods, and Applications*, 2nd ed.; SIAM: Philadelphia, PA, USA, 2014. [CrossRef]
25. Vansteenwegen, P.; Souffriau, W.; Van Oudheusden, D. The orienteering problem: A survey. *Eur. J. Oper. Res.* **2011**, *209*, 1–10. [CrossRef]
26. Bani Ahmad, A.Y.A.; Verma, N.; Sarhan, N.M.; Awwad, E.M.; Arora, A.; Nyangaresi, V.O. An IoT and blockchain-based secure and transparent supply chain management framework in smart cities using optimal queue model. *IEEE Access* **2024**, *12*, 51752–51771. [CrossRef]
27. Ivanov, D.; Dolgui, A. A digital supply chain twin for managing the disruption risks and resilience in the era of Industry 4.0. *Prod. Plan. Control* **2021**, *32*, 775–788. [CrossRef]
28. Kayvanfar, V.; Elomri, A.; Kerbache, L.; Vandchali, H.R.; El Omri, A. A review of decision support systems in the internet of things and supply chain and logistics using web content mining. *Supply Chain Anal.* **2024**, *6*, 100063. [CrossRef]
29. Hussain, M.; Javed, W.; Hakeem, O.; Yousafzai, A.; Younas, A.; Awan, M.J.; Nobanee, H.; Zain, A.M. Blockchain-based IoT devices in supply chain management: A systematic literature review. *Sustainability* **2021**, *13*, 13646. [CrossRef]
30. Dutta, P.; Chavhan, R.; Gowtham, P.; Singh, A. The individual and integrated impact of blockchain and IoT on sustainable supply chains: A systematic review. *Supply Chain Forum Int. J.* **2023**, *24*, 103–126. [CrossRef]
31. Idrissi, Z.K.; Lachgar, M.; Hrimech, H. Blockchain, IoT and AI in logistics and transportation: A systematic review. *Transp. Econ. Manag.* **2024**, *2*, 275–285. [CrossRef]
32. Tiwari, M.K.; Bidanda, B.; Geunes, J.; Fernandes, K.; Dolgui, A. Supply chain digitisation and management. *Int. J. Prod. Res.* **2024**, *62*, 2918–2926. [CrossRef]
33. Dong, Z.; Liang, W.; Liang, Y.; Gao, W.; Lu, Y. Blockchained supply chain management based on IoT tracking and machine learning. *EURASIP J. Wirel. Commun. Netw.* **2022**, *2022*, 127. [CrossRef] [PubMed]
34. Hu, H.; Xu, J.; Liu, M.; Lim, M.K. Vaccine supply chain management: An intelligent system utilizing blockchain, IoT and machine learning. *J. Bus. Res.* **2023**, *156*, 113480. [CrossRef] [PubMed]
35. Kumar, D.; Singh, R.K.; Mishra, R.; Daim, T.U. Roadmap for integrating blockchain with Internet of Things (IoT) for sustainable and secured operations in logistics and supply chains: Decision making framework. *Technol. Forecast. Soc. Change* **2023**, *196*, 122837. [CrossRef]
36. Kalganova, T.; Dzalbs, I. *Supply Chain Logistics Problem Dataset*; Brunel University London: London, UK, 2019. [CrossRef]
37. Dzalbs, I.; Kalganova, T. Accelerating supply chains with Ant Colony Optimization across a range of hardware solutions. *Comput. Ind. Eng.* **2020**, *147*, 106610. [CrossRef] [PubMed]
38. Castro, M.; Liskov, B. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.* **2002**, *20*, 398–461. [CrossRef]
39. Zilberstein, S. Using anytime algorithms in intelligent systems. *AI Mag.* **1996**, *17*, 73–83. [CrossRef]
40. Benmamoun, Z.; Khlie, K.; Agarwal, V.; Kaul, A.; Sharma, R.; Navaneetha Krishnan, S. Sustainable supply chain strategy under carbon trading. *Bus. Strategy Environ.* 2026; *early view*. [CrossRef]

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.