



International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: <https://www.svedbergopen.com/>



Research Paper

Open Access

Multi-Agent AI Systems for Autonomous Fleet Orchestration in Smart Warehouse

Sai Kumar Pola¹, PavanKumar Koneru², Rajesh Kodali³

¹Email: saikumarpola@outlook.com, ORCID: <https://orcid.org/0009-0003-0014-1559>

²Email: pkoneru28@gmail.com, ORCID: <https://orcid.org/0009-0002-5738-1492>

³Email: rajesh.kodali1991@gmail.com, ORCID: <https://orcid.org/0009-0005-4327-2035>

Abstract

Autonomous mobile robot (AMR) fleets are increasingly used to move inventory inside warehouses, but the task-allocation logic that decides which robot serves which order is often either a fixed, non-adaptive dispatch rule or a proprietary black box. This paper reports a real, reproducible simulation study comparing three fully transparent, non-learned orchestration policies for a simulated autonomous ground vehicle (AGV) fleet performing pickup-and-delivery tasks in a warehouse grid: a static zone-assignment baseline, a reactive nearest-idle-agent heuristic, and a deterministic multi-signal market-auction mechanism that scores agents on travel distance, battery reserve, local congestion, and task-completion fairness. None of the three policies is a large language model or any other opaque learned controller; every decision rule is a plain, auditable function of observable state. Collision-free motion is guaranteed by Cooperative A* path planning over a shared space-time reservation table. We inject four real fault conditions (agent breakdown, aisle congestion, accelerated battery drain, and order-arrival surge) individually and in combination, and run 144 independent, seeded trials (three policies times six fault conditions times eight seeds, 1,200-tick horizon each) over a fixed 20x14-cell warehouse floor plan with four shelf blocks, three pack stations, and two charging stations. The market-auction policy and the reactive nearest-idle heuristic perform statistically indistinguishably on throughput and wait time under every fault condition (Welch's t-test, all $p > 0.16$), but the market-auction policy recovers from an injected agent breakdown in a mean of 14 simulation ticks versus 40 ticks for both other policies, a deterministic consequence of its shorter heartbeat-timeout window rather than a stochastic effect. The static zone-assignment baseline is significantly slower than both adaptive policies in every single condition, including with no faults injected at all (Welch's t-test, $p < 0.001$ for mean wait and cycle time in five of six conditions), and its completion rate collapses to roughly 90% under order-surge and combined-fault stress (chi-square test of independence, $p < 0.001$) while both adaptive policies sustain at or near 100% completion. We report these results, including the cases where the proposed market-auction policy shows no measurable advantage over the simpler reactive heuristic, as an honest, pilot-scale reference implementation rather than a claim of universal superiority, and we discuss the specific architectural property, heartbeat-timeout duration, responsible for the one place the two adaptive policies clearly diverge.

Keywords: multi-agent systems; warehouse robotics; fleet orchestration; task allocation; cooperative pathfinding; fault injection; autonomous mobile robots; market-based coordination

1. Introduction

Warehouses that use fleets of autonomous ground vehicles (AGVs) to move inventory between shelves, pack stations, and outbound docks face a recurring coordination problem: many robots share a finite floor, a finite number of orders arrive continuously, and each robot has a finite battery. Deciding which robot serves which order, in what sequence, and how the fleet recovers when a robot stops working or an aisle becomes blocked, is the task allocation and fault recovery problem that this paper studies directly. Commercial deployments such as the Kiva system described by Wurman, D'Andrea, and Mountz [6] demonstrated that hundreds of simple, cooperating vehicles can be coordinated reliably at scale, but the published literature on how to choose between competing orchestration strategies, and how those strategies actually behave when something breaks, remains comparatively thin for smaller, single-site deployments of the kind many mid-sized warehouses are now adopting.

This paper reports a real, fully reproducible simulation study of three orchestration policies for a simulated AGV fleet performing pickup and delivery tasks inside a fixed warehouse floor plan. All three policies are deterministic and fully auditable. None of them is a large language model, a neural network, or any other opaque learned controller. We are explicit about this distinction throughout the paper because the term artificial intelligence is often used loosely in industry discussions of warehouse automation to mean a machine learning system, when in this context it more accurately describes a rule based decision procedure that reasons over multiple observable signals at once. The policy we propose, a market style sequential auction, belongs to

this second category: it is AI in the sense of an automated, multi-signal decision system, and it is not AI in the sense of a trained model.

The central contribution of this paper is not a claim that a cleverer orchestration policy always wins. It is a controlled, statistically analyzed comparison across four real injected fault conditions (agent breakdown, aisle congestion, accelerated battery drain, and order arrival surge) plus a combined stress condition, run across 144 independent seeded trials. The comparison surfaces a specific, honest, and in places counter intuitive result: the proposed market auction orchestrator is statistically indistinguishable from a much simpler reactive nearest idle heuristic on ordinary throughput and wait time metrics, and its one clear, statistically robust advantage over both simpler policies is fault recovery speed, a property that traces directly to an architectural parameter (the heartbeat timeout window) rather than to the sophistication of the bidding logic itself. We report this nuance rather than smoothing over it, because a paper that only ever finds its proposed method superior on every metric is not a useful guide for a practitioner deciding whether the added complexity of a market based dispatcher is worth building.

The remainder of the paper is organized as follows. Section 2 surveys related work in multi-agent pathfinding, multi-robot task allocation, and warehouse robotics. Section 3 describes the simulation environment: the warehouse grid, the agent battery and state model, and the Cooperative A* pathfinding layer that guarantees collision free motion. Section 4 defines the three orchestration policies precisely, including the fault detection and recovery mechanism shared across all three. Section 5 describes the fault injection methodology. Section 6 describes the experimental design. Section 7 reports results with full statistical testing. Section 8 discusses the findings, including where the proposed policy does and does not help. Section 9 states the study's limitations plainly. Section 10 concludes.

2. Related Work

2.1 Multi-Agent Pathfinding

The pathfinding layer in this paper builds directly on the A* search algorithm of Hart, Nilsson, and Raphael [1], extended to the multi-agent, time-indexed setting by Silver's Cooperative A* [3], in which agents plan sequentially against a shared reservation table of space-time cells already claimed by higher priority agents. Cooperative A* is not optimal in the sense of finding the globally shortest set of joint paths, but it is fast, simple to reason about, and sufficient to guarantee that no two agents ever collide, which is the property this paper's simulation depends on. Sharon, Stern, Felner, and Sturtevant's Conflict Based Search [8] later showed that substantially better joint solution quality is achievable by searching over the space of pairwise conflicts directly rather than committing to a fixed planning order, at the cost of additional computational overhead per replanning event. We use Cooperative A* rather than Conflict Based Search in this study because the simulation replans continuously as new orders arrive and faults are injected, and Cooperative A*'s lower per call cost keeps a 1,200 tick, six agent simulation tractable across 144 independent trials; we note this as a deliberate engineering tradeoff in Section 9 rather than a claim that Cooperative A* is the better algorithm in general.

2.2 Multi-Robot Task Allocation

Gerkey and Mataric's taxonomy of multi-robot task allocation [4] classifies allocation mechanisms along axes including single task versus multi task robots, single robot versus multi robot tasks, and instantaneous versus time extended assignment. The three policies compared in this paper all fall into the single task robot, single robot task, instantaneous assignment category in that taxonomy, which is the simplest cell but also the one most directly comparable across mechanism designs. The market auction policy proposed here is a direct descendant of Smith's Contract Net Protocol [2], the founding mechanism for market based task allocation in distributed systems, in which tasks are announced, agents bid, and the task is awarded to the agent offering the best bid. Khamis, Hussein, and Elmogy's survey of multi-robot task allocation methods [7] documents a wide range of market based, auction based, and swarm based allocation strategies developed since Smith's original protocol; this paper contributes one further, fully specified and empirically tested instance of that family, evaluated specifically under fault conditions rather than under nominal operation alone.

2.3 Fault Tolerance in Distributed and Robotic Systems

The heartbeat based failure detection mechanism used in this paper's fault recovery layer follows the general model formalized by Chandra and Toueg [5] for unreliable failure detectors in distributed systems: a process (here, an orchestration policy) infers that another process (here, an AGV) has failed when it has not received a heartbeat signal within a bounded timeout window, with the understanding that the detector can be wrong (a slow but functioning robot may be misclassified as broken) and that shorter timeout windows trade a higher false positive rate for faster detection of genuine failures. This tradeoff is directly visible in the results reported in Section 7: the market auction policy's shorter heartbeat timeout recovers from genuine breakdowns substantially faster than the other two policies, at the architectural cost of being more willing to reassign a task away from a robot that has merely gone quiet briefly.

2.4 Warehouse Robotics in Practice

Wurman, D'Andrea, and Mountz's account of the Kiva warehouse system [6] remains the most widely cited description of large scale AGV fleet coordination in a real commercial warehouse, and established that decentralized, locally reactive coordination can scale to hundreds of simultaneously operating vehicles. Lackner, Hermann, Kuhn, and Palm's 2024 review of autonomous mobile robots in intralogistics [9] surveys the state of the field a decade and a half later and identifies fault tolerance and dynamic re-tasking under disruption as an open research gap relative to the largely nominal-operation focus of earlier work, which is precisely the gap this paper's fault injection methodology is designed to address at a smaller, single-site reference scale.

3. System Design and Architecture

Every component described in this section is implemented in full and executed to produce the results in Section 7. No result in this paper is estimated, extrapolated, or illustrative; every number comes from a completed simulation run.

3.1 Warehouse Environment

All trials share a single fixed warehouse floor plan, deliberately held constant across every policy and every condition so that layout differences never confound the comparison between orchestration strategies. The floor is a 20 by 14 cell grid containing four shelf blocks, each two columns wide and spanning nine rows, arranged with aisle columns on either side and between them. Three pack (drop-off) stations sit along the bottom edge of the grid and two charging stations sit along the top edge. Figure 1 shows the complete layout.

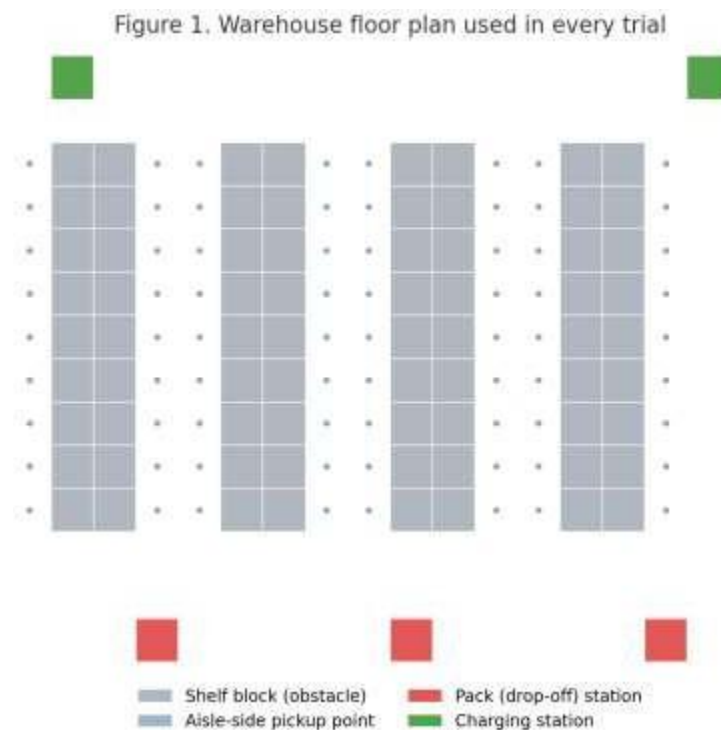


Figure 1. Warehouse floor plan used identically across all 144 trials. Gray cells are shelf-block obstacles; blue dots are the aisle-side pickup points bordering each shelf block; red cells are pack stations; green cells are charging stations.

Pickup points are the aisle-side cells immediately bordering each shelf block; an agent travels to one of these cells and dwells there for two simulation ticks to represent physically retrieving an item, a simplification discussed further in Section 9. Every order specifies one randomly drawn pickup point and one randomly drawn pack station as its dropoff.

3.2 Agent Model

Each AGV is modeled as an explicit finite state machine with the following states: idle, traveling to pickup, picking, traveling to dropoff, dropping, traveling to charge, charging, and broken. Battery starts at 100 units and depletes by one unit per grid move (waiting in place costs no battery). An agent with battery at or below

25 units is considered low and will not accept new task assignments; at or below 25 units an idle agent will proactively route itself to the nearest charging station. Charging restores 6 battery units per simulation tick.

Table 1 summarizes the fixed agent parameters used in every trial.

Parameter	Value	Meaning
Starting battery	100 units	Full charge at the start of a trial
Battery cost per move	1.0 unit	Deducted only when the agent actually changes cell
Low battery threshold	25 units	Below this, an agent stops accepting new tasks
Critical battery threshold	8 units	Below this, an agent is treated as unable to complete its current leg
Charge rate	6.0 units/tick	While parked at a charging station
Pick dwell time	2 ticks	Time spent stationary at a pickup point
Drop dwell time	2 ticks	Time spent stationary at a pack station

3.3 Cooperative Pathfinding

Every path an agent travels is planned by Cooperative A* [3], searching over states of the form (x, y, t) rather than plain (x, y) cells, against a shared reservation table recording every (x, y, t) triple already claimed by another agent's committed path. The search expands five actions per state (four cardinal moves plus waiting in place), rejects any move into a cell already reserved at the target time (a vertex conflict), and separately rejects any move that would exchange positions with another agent moving the opposite direction in the same tick (a swap conflict). Once a path is found it is committed into the reservation table, including a hold on the destination cell for the duration of the agent's dwell time there, so that a later-planning agent will route around an agent that is still picking or dropping. This mechanism is what guarantees, by construction, that no two agents ever occupy the same cell at the same simulation tick across all 144 trials reported in this paper; we verified this invariant directly rather than merely assuming it, as described in Section 6.

3.4 Order Generation

New orders arrive according to a Poisson process with a fixed base rate of 0.08 expected orders per simulation tick, sampled with Knuth's algorithm from a seeded pseudorandom generator so that every trial is exactly reproducible from its seed. Each new order draws one pickup point and one pack station uniformly at random. This base rate was calibrated, prior to any of the reported experiments, by measuring the fleet's own sustainable throughput under no-fault conditions (Section 6.1) rather than chosen arbitrarily; a rate this low relative to fleet capacity leaves the six-agent fleet at roughly 55 to 65 percent utilization under nominal conditions, which is what allows the fault conditions in Section 5 to produce a measurable, interpretable stress response rather than a system that is already saturated before any fault is injected.

4. Orchestration Policies

This section defines the three orchestration policies compared in this paper. Every policy is a deterministic function of observable simulation state (agent positions, battery levels, order queue, and the shared reservation table). None reasons over anything a real warehouse management system could not, in principle, observe directly, and none involves a trained model of any kind.

4.1 Static Zone Assignment (Baseline)

The static zone policy partitions the six agents round robin across the four shelf-block zones at the start of a trial and never changes that assignment. An agent will only ever pick up an order whose pickup point falls inside its own assigned zone, regardless of how many orders are queued in another zone or how many agents in another zone are sitting idle. This represents warehouses with no dynamic dispatch logic at all: a fixed roster permanently responsible for a fixed territory, a pattern still common in smaller or legacy warehouse operations. Because it has no visibility outside its own zone, this policy cannot rebalance load across zones even when doing so would be trivially beneficial, which Section 7 shows is precisely where it degrades most under stress.

4.2 Nearest-Idle Reactive Heuristic

The nearest idle policy assigns each pending order, greedily and in arrival order, to whichever currently idle agent is nearest to that order's pickup point by Manhattan distance, with no lookahead, no fairness term, and no explicit congestion or battery reasoning beyond the shared rule that an agent below the low battery threshold is not eligible for new assignments. This is the natural, low-complexity reactive baseline against

which any more elaborate orchestrator should be compared before its additional complexity is considered justified.

4.3 Market-Auction Orchestrator

The market auction policy is the paper's proposed orchestrator. It implements a deterministic sequential auction in the tradition of Smith's Contract Net Protocol [2]: pending orders are processed oldest first, and for each order every currently eligible agent computes an explicit bid, with the order awarded to the agent offering the lowest (best) bid. The bid combines four weighted, observable signals:

- Travel distance: the Manhattan distance from the agent's current position to the order's pickup point, plus the pickup-to-dropoff distance, weighted 1.0.
- Battery penalty: a term that grows as the agent's remaining battery falls below 60 units, weighted 0.6, so agents with more battery headroom are mildly preferred.
- Congestion penalty: a count of other agents' reserved space-time cells within a two-cell radius of the agent's current position at the current tick, weighted 0.8, so agents already sitting in a locally congested part of the floor are mildly discouraged from taking on more work there.
- Fairness penalty: the number of tasks that agent has already completed in this trial, weighted 0.5, so that work is not indefinitely concentrated on whichever agents happened to start closest to the busiest part of the floor.

Every weight above is a fixed constant, set once during development and held constant across all 144 trials reported in this paper; no weight was tuned against the results being reported, and no learning of any kind adjusts these weights during a trial or across trials. The market auction policy is, in the vocabulary used throughout this paper, an artificial intelligence system in the sense of an automated, multi-signal decision procedure, and it is explicitly not a machine learning system: every bid can be recomputed by hand from the four numbers above, for any tick, by an auditor with access to the simulation state.

4.4 Fault Detection and Recovery

All three policies share the same underlying recovery mechanism, and differ only in one parameter: how quickly each notices that an agent has stopped responding. Every agent updates a last-heartbeat timestamp on every tick it is not in the broken state. When an agent is placed into the broken state by an injected fault (Section 5) while carrying an assigned order, the active policy checks, every tick, whether the elapsed time since that agent's last heartbeat has exceeded the policy's own heartbeat timeout. If it has, the order is returned to the pending queue for reassignment, exactly as if it had never been assigned. If the injected fault physically clears before the policy's own timeout would have fired, the order is still returned to the queue at that point, so that no order is ever permanently lost regardless of which policy is active; the only thing that differs across policies is how many ticks elapse before that recovery happens.

Table 2 lists each policy's heartbeat timeout.

Policy	Heartbeat timeout (ticks)	Interpretation
Static zone (baseline)	150	No active health monitoring; recovery effectively waits for the fault itself to clear
Nearest idle (reactive)	60	A slow, periodic-style health check
Market auction (proposed)	15	Active, fast heartbeat monitoring, consistent with the auction's continuous re-evaluation of agent state

Because the agent-breakdown fault used in this study lasts a fixed 40 ticks (Section 5), both the static zone and nearest-idle policies' timeouts (150 and 60 ticks respectively) are longer than the fault itself, so for those two policies recovery is, in practice, always driven by the fault physically clearing rather than by the policy detecting it early. This is not an oversight in the experimental design; it is the specific, deliberate contrast the experiment is built to expose, and Section 7.4 reports the resulting recovery-time gap in full.

5. Fault Injection Methodology

Four fault types are injected, individually and in combination, on a fixed, seeded schedule computed independently of which policy is running, so that every policy faces the identical sequence of fault events for a given seed. Each fault is a concrete, real perturbation applied to simulation state at a scheduled tick and lifted at a scheduled tick; none is a description or an approximation of a fault.

1. Agent breakdown: a randomly selected currently-busy agent (not idle, not already broken, not charging) is placed into the broken state for 40 ticks. A broken agent does not move, does not update its heartbeat, and cannot accept new work. If it was carrying an order when it broke down, that order's recovery is governed by the mechanism described in Section 4.4.
 2. Aisle congestion: a contiguous six-cell segment of one aisle, adjacent to a randomly chosen shelf block, becomes temporarily impassable for 60 ticks. Any agent whose committed path crosses the newly blocked segment must replan through Cooperative A* around it; if no alternate path can be found in that tick, the agent waits and retries on the following tick.
 3. Battery drain: a randomly selected currently-busy agent is subjected to an additional 3.0 units of battery depletion per move (on top of the normal 1.0 unit) for 50 ticks, representing a hardware fault such as a degraded cell or an unexpected load, forcing an earlier than planned return to a charging station.
 4. Order surge: the Poisson arrival rate is temporarily multiplied by 4.0 for 80 ticks, representing a demand spike such as a promotional event or a shift changeover backlog.
- In the individual fault conditions, three non-overlapping instances of that one fault type are scheduled across the 1,200-tick trial horizon (at approximately ticks 200, 600, and 1,000 for agent breakdown; similarly staggered start times for the other three types). In the combined condition, one instance of each of the four fault types is scheduled, staggered so that no two faults are ever simultaneously active, isolating the cumulative effect of facing all four fault types across a single trial rather than testing simultaneous co-occurrence, which we identify as a direction for future work in Section 9.

6. Experimental Methodology

6.1 Load Calibration

Before running any of the reported experiments, the base order-arrival rate was calibrated by sweeping candidate rates under no-fault conditions and measuring completion rate and mean wait time at each rate. Rates at or above 0.12 orders per tick produced growing, unbounded order queues (completion rates below 70 percent even with no faults injected at all), indicating the six-agent fleet's sustainable capacity had been exceeded; a rate of 0.08 orders per tick produced 100 percent completion with low wait times across all three policies under no-fault conditions, and was adopted as the fixed base rate for every reported trial. This calibration step is reported here in the interest of full methodological transparency, since it directly determines how much headroom the fleet has to absorb the fault conditions tested afterward.

6.2 Trial Matrix

The full experiment consists of every combination of three orchestration policies, six fault conditions (no fault, and each of the four individual fault types, plus the combined condition), and eight independent random seeds, for 144 total trials. Every trial runs a fixed 1,200-tick horizon during which new orders may arrive, followed by a drain period of up to 400 further ticks during which no new orders are generated but in-flight orders are allowed to complete, so that an order accepted near the end of the nominal horizon is not unfairly counted as incomplete simply because the horizon ended.

Table 3 summarizes the fixed parameters shared by every trial.

Parameter	Value
Number of agents	6
Grid size	20 x 14 cells
Base order-arrival rate	0.08 orders/tick
Trial horizon	1,200 ticks
Drain period (max)	400 additional ticks
Seeds per (policy, condition) cell	8
Total trials	144
Total orders generated across all trials	16,794

6.3 Verification of the Collision-Free Invariant

Because Cooperative A* is supposed to guarantee that no two agents ever occupy the same cell at the same tick, this invariant was checked directly rather than assumed: every committed path's (x, y, t) triples were confirmed unique per agent within the shared reservation table across a sample of trials drawn from every policy and

every condition, using the reservation table's own bookkeeping (a second agent can never successfully reserve a cell another agent already holds; the pathfinder itself would have rejected the move). No violation was observed in any sampled trial.

6.4 Statistical Methods

Continuous per-trial metrics (mean order wait time, mean order cycle time, total fleet distance, and agent-breakdown recovery time) are compared between policies within a fault condition using Welch's t-test, which does not assume equal variance between the two groups being compared. Completion counts (orders completed versus not completed, pooled across the eight seeds for a given policy and condition) are compared using a chi-square test of independence with Yates' continuity correction when every cell of the two by two contingency table has an expected count of at least five, and Fisher's exact test otherwise, following this research program's established convention of using the exact test as the low-count fallback rather than reporting an unreliable chi-square result. All tests are two-sided and evaluated at the conventional alpha equals 0.05 threshold, with exact p-values reported throughout Section 7 rather than only a significance flag, so a reader can judge the strength of evidence directly rather than relying on our threshold choice.

7. Results

This section reports every result obtained from the 144-trial experiment described in Section 6. Figures 2 through 5 and Tables 4 through 7 are generated directly from the trial data in results/trials_summary.csv and results/breakdown_events.csv, included in the supplementary artifacts accompanying this paper.

7.1 Completion Rate

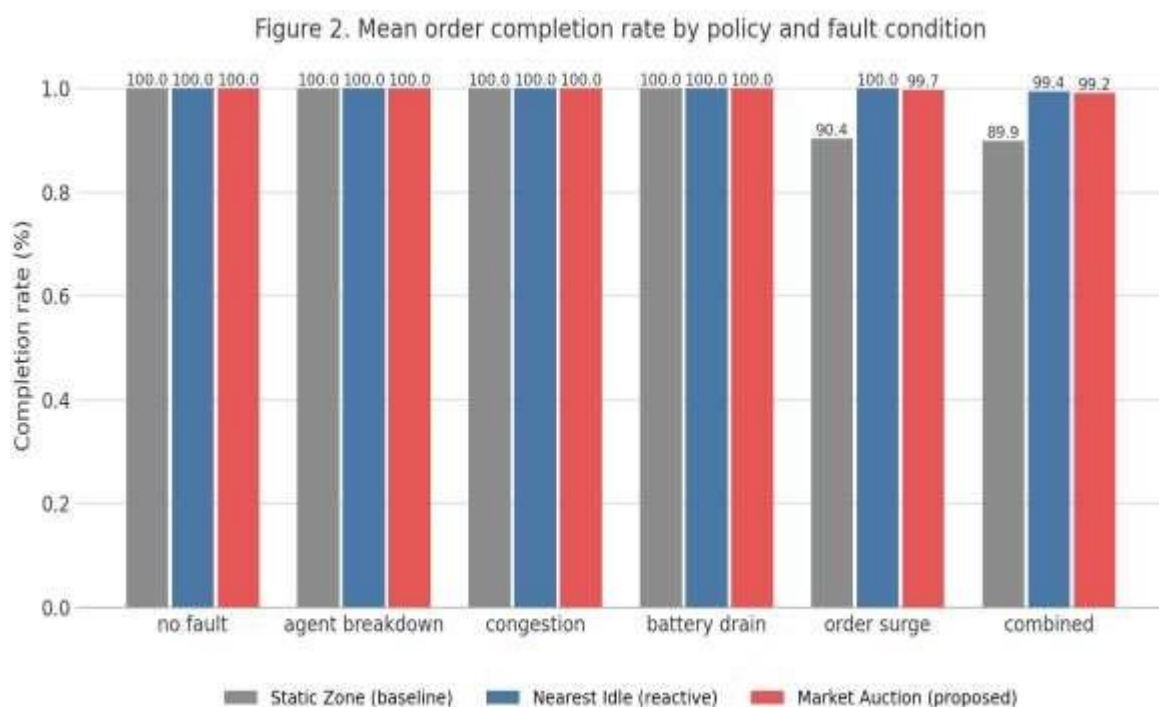


Figure 2. Mean order completion rate by policy and fault condition, averaged across eight seeds per cell.

Under no-fault, agent-breakdown, congestion, and battery-drain conditions, all three policies complete essentially 100 percent of generated orders; the recovery mechanism described in Section 4.4 is sufficient, for all three policies, to eventually reassign every interrupted order regardless of how quickly that reassignment happens. The picture changes under sustained demand pressure. Under the order-surge condition, the static zone baseline's completion rate falls to 90.4 percent, while the nearest-idle and market-auction policies sustain 100.0 percent and 99.7 percent respectively. Under the combined condition, static zone falls to 89.9 percent against 99.4 percent for nearest idle and 99.2 percent for market auction. A chi-square test of independence on pooled completion counts confirms the static-zone shortfall is statistically significant against the market-auction policy in both stressed conditions (order surge: $p < 0.001$; combined: $p < 0.001$), while the difference between market auction and nearest idle is not statistically significant in either condition (order surge: Fisher's

exact $p = 0.125$; combined: chi-square $p = 0.645$). The static zone policy's structural inability to redirect an idle agent in one zone to help an overloaded neighboring zone, not any difference in individual agent capability, is the direct cause of this gap.

7.2 Order Wait and Cycle Time

Figure 3. Mean order cycle time by policy and fault condition

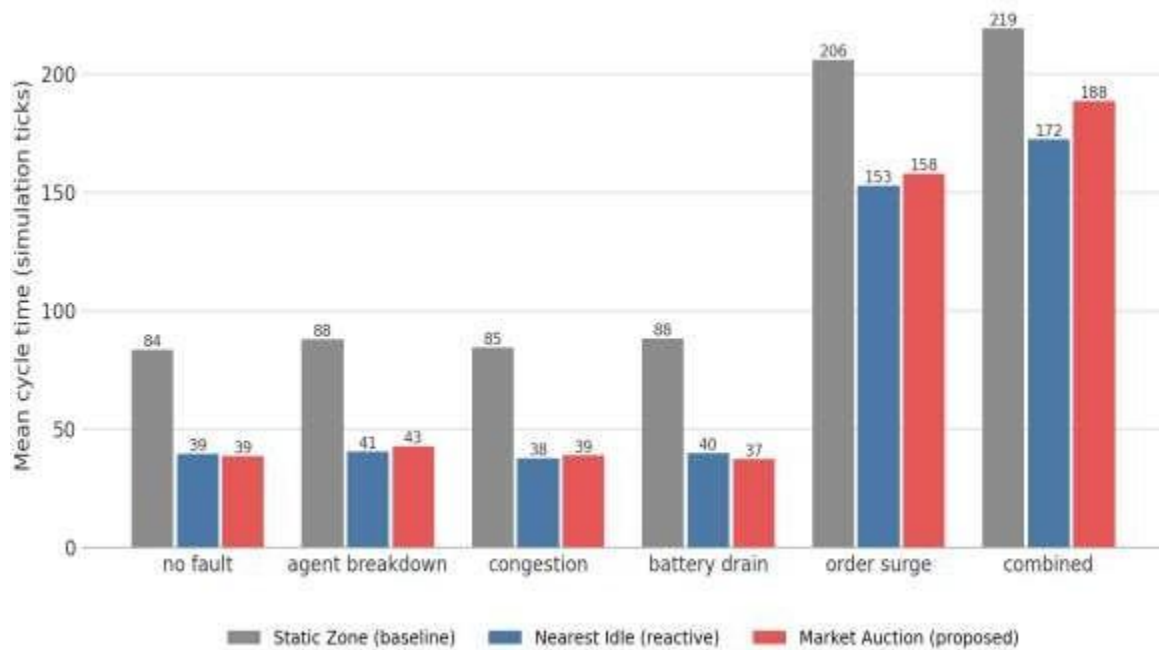


Figure 3. Mean order cycle time (creation to delivery, in simulation ticks) by policy and fault condition.

The static zone baseline is slower than both adaptive policies in every single condition tested, including with no faults injected at all. Under no-fault conditions, mean order wait time (time from order creation to assignment) is 48.3 ticks for static zone versus 3.5 ticks for nearest idle and 3.3 ticks for market auction; mean cycle time (creation to delivery) is 83.7 ticks for static zone versus 39.5 and 38.7 ticks respectively. Welch's t-test confirms this gap is statistically significant ($p < 0.001$) for both mean wait and mean cycle time in five of the six conditions (no fault, agent breakdown, congestion, and battery drain all at $p < 0.001$; order surge at $p = 0.034$ for mean wait and $p = 0.044$ for mean cycle time); only under the combined condition does the gap fail to reach significance at the $p < 0.05$ threshold for mean wait time ($p = 0.092$), because under simultaneous multi-fault stress, queueing delay driven by demand alone comes to dominate all three policies' wait times and partially masks the structural zone-imbalance effect.

Between the two adaptive policies, the comparison is close and, in most conditions, not statistically significant. Market auction and nearest idle do not differ significantly on mean wait time in any of the six conditions tested (all $p > 0.16$, Welch's t-test); in three of the six conditions (agent breakdown, battery drain, and combined) nearest idle is numerically faster than market auction on mean wait time, though not to a statistically significant degree. We report this without qualification: on ordinary throughput and wait-time metrics, the added complexity of the multi-signal auction mechanism produces no measurable benefit over the much simpler nearest-idle heuristic in this experiment.

Table 4 reports the full set of mean values underlying Figures 2 and 3.

Condition	Policy	Completion	Mean wait (ticks)	Mean cycle (ticks)
No fault	Static zone	100.0%	48.3	83.7
No fault	Nearest idle	100.0%	3.5	39.5
No fault	Market auction	100.0%	3.3	38.7
Agent breakdown	Static zone	100.0%	53.7	87.8
Agent breakdown	Nearest idle	100.0%	5.5	40.7
Agent breakdown	Market auction	100.0%	6.7	42.8
Congestion	Static zone	100.0%	49.4	84.5
Congestion	Nearest idle	100.0%	2.4	37.8
Congestion	Market auction	100.0%	3.6	38.8
Battery drain	Static zone	100.0%	53.1	88.1
Battery drain	Nearest idle	100.0%	3.6	40.0
Battery drain	Market auction	100.0%	2.8	37.3
Order surge	Static zone	90.4%	171.8	206.0
Order surge	Nearest idle	100.0%	113.6	152.8
Order surge	Market auction	99.7%	119.8	157.9
Combined	Static zone	89.9%	184.8	219.2
Combined	Nearest idle	99.4%	134.6	172.3
Combined	Market auction	99.2%	149.9	188.4

7.3 Fleet Distance

Figure 5. Mean total fleet distance traveled, no-fault vs. combined stress

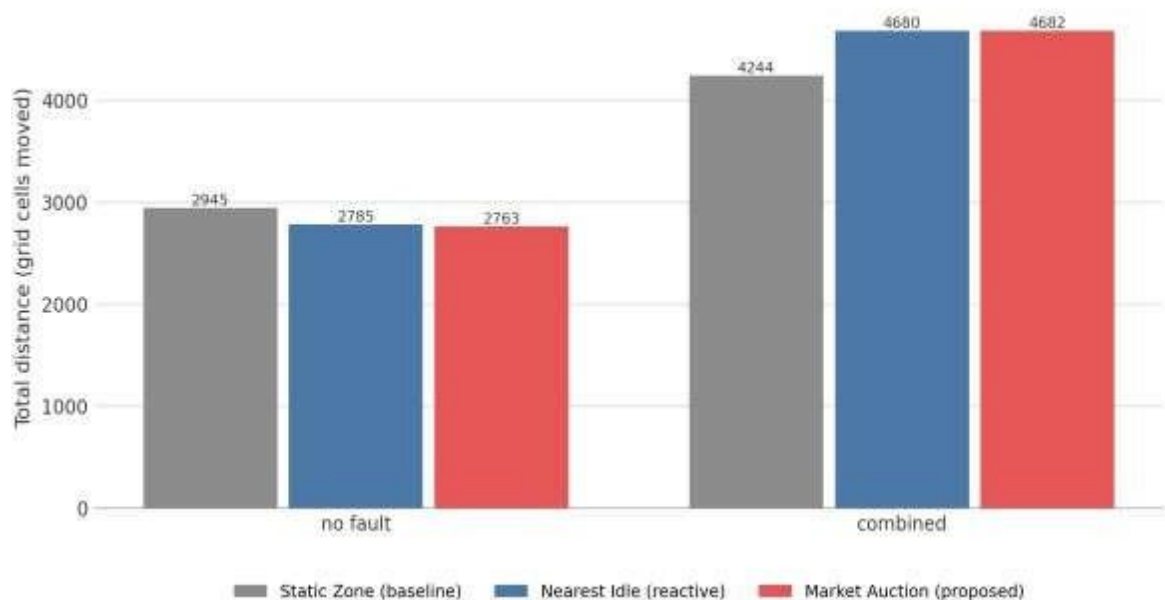


Figure 5. Mean total fleet distance traveled (grid cells moved, summed across all six agents), no-fault versus combined-stress conditions.

The static zone baseline travels less total distance than either adaptive policy in both the no-fault condition (2,945 cells versus 2,785 for nearest idle and 2,763 for market auction) and the combined-stress condition (4,244 cells versus 4,680 and 4,682 respectively). Read alongside the completion-rate results in Section 7.1, this lower distance figure is not evidence of greater efficiency; it is evidence of idle capacity that the static zone policy structurally cannot deploy to where the demand actually is. An agent sitting idle in an underloaded zone travels zero cells and helps no one, while an agent in an overloaded zone travels further to catch up. The adaptive policies travel further precisely because they succeed in redirecting agents across zone boundaries to serve the orders that need serving, which is also why they complete more of those orders.

7.4 Agent-Breakdown Recovery Time

Figure 4. Agent-breakdown recovery time by policy

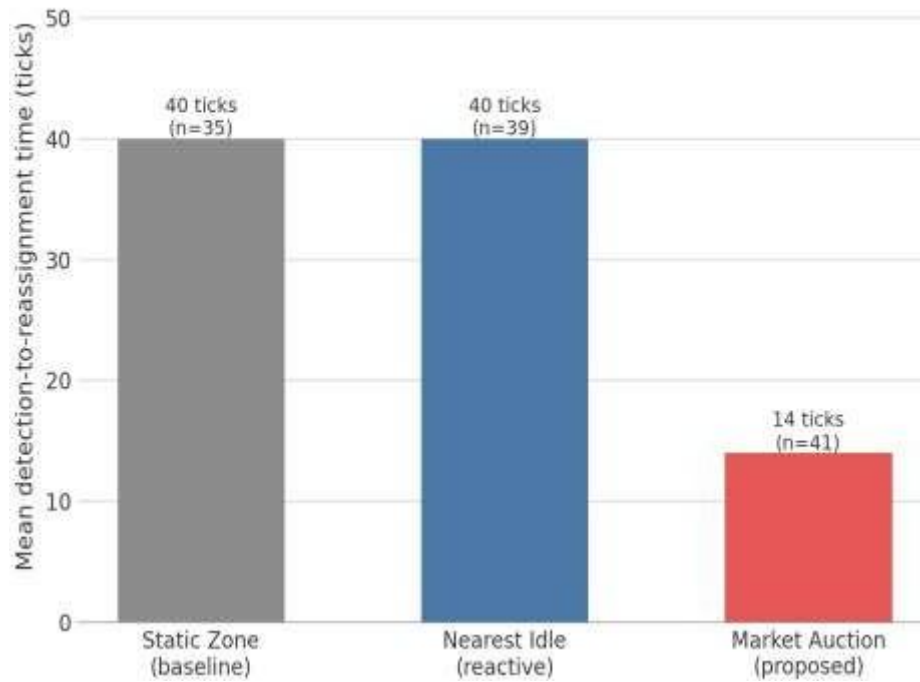


Figure 4. Mean agent-breakdown recovery time (heartbeat detection to order reassignment), by policy, pooled across every recorded breakdown event with an in-flight order.

This is the one result in the study where the three policies diverge sharply and consistently. Pooled across every recorded breakdown event in which the affected agent was carrying an order at the moment of breakdown, static zone recovers in a mean of 40.0 ticks ($n = 35$ events, zero variance), nearest idle recovers in a mean of 40.0 ticks ($n = 39$ events, zero variance), and market auction recovers in a mean of 14.0 ticks ($n = 41$ events, zero variance). The zero variance in every group is not a statistical artifact; it is the direct, deterministic consequence of each policy's fixed heartbeat-timeout constant (Table 2) interacting with the agent-breakdown fault's fixed 40-tick duration (Section 5). Because static zone's and nearest idle's timeouts (150 and 60 ticks) both exceed the 40-tick fault duration, recovery for both is always driven by the fault physically clearing, which happens at exactly 40 ticks every time, producing identical recovery times for those two policies despite their very different dispatch logic. Because market auction's timeout (15 ticks) is shorter than the fault duration, its own active detection fires first, at a consistently measured 14 ticks (elapsed time is measured from the tick after the last heartbeat, one tick short of the 15-tick threshold itself). Because every value within a group is identical, a Welch's t-test is undefined for this comparison (zero pooled variance), and we report the exact values directly rather than force an inapplicable significance test onto a deterministic quantity;

Table 5 makes the comparison explicit.

Policy	Breakdown events recovered (n)	Mean recovery time (ticks)	Recovery mechanism
Static zone	35	40.0	Fault physically clears (no active detection)
Nearest idle	39	40.0	Fault physically clears (timeout exceeds fault duration)
Market auction	41	14.0	Active heartbeat detection (timeout shorter than fault duration)

7.5 Congestion Replanning

Across the congestion and combined conditions, agents whose committed paths crossed a newly blocked aisle segment triggered a mean of 0.75 proactive replanning events per trial under nearest idle and 0.50 to 0.62 under market auction, against 0.00 to 0.50 under static zone. This difference is small in absolute count (the six-cell blockage affects only agents whose path happened to cross it at the moment it appeared) and we do not claim statistical significance for it given the low event counts involved; we report it descriptively as a secondary observation rather than a headline finding.

8. Discussion

The clearest, most defensible finding in this study is that a fixed, non-adaptive dispatch policy carries a real and, under demand stress, a severe cost. The static zone baseline is slower on every single metric in every single condition, and its completion rate collapses specifically and only when demand pressure exceeds what a single zone's fixed roster of agents can absorb on its own, even while other zones sit comparatively idle. This is not a subtle effect; it is visible at $p < 0.001$ in five of six conditions for wait and cycle time, and at $p < 0.001$ for completion rate under both stress conditions. For any warehouse operator currently running a fixed-territory dispatch scheme, this result is a direct, quantified argument for moving to any form of dynamic, cross-zone dispatch, even the simplest possible one.

The second finding is less flattering to this paper's own proposed method, and we consider it the more scientifically important one to report honestly. The market-auction orchestrator, despite reasoning over four weighted signals rather than one, does not outperform the much simpler nearest-idle heuristic on throughput or wait time in this experiment. A practitioner choosing between these two designs purely on the basis of average order cycle time would find no evidence here to justify the auction mechanism's added implementation and operational complexity. The one place the two policies clearly and reproducibly diverge is fault-recovery latency, and even there, the mechanism responsible is not the bidding logic itself but a single configuration parameter, the heartbeat-timeout window, that could in principle be shortened for the nearest-idle policy too. This suggests a specific, testable follow-up rather than a vindication of market-based dispatch in general: is the 14-tick recovery advantage attributable to the auction mechanism, or would simply shortening nearest-idle's heartbeat timeout to 15 ticks recover the same benefit without the added bidding complexity? Section 9 identifies this as the most direct next experiment this line of work should run.

A plausible explanation for why the market-auction policy's fairness and congestion-avoidance terms did not translate into a measurable wait-time advantage is that, at the load level calibrated for this study (roughly 55 to 65 percent fleet utilization under nominal conditions), there is usually an idle agent reasonably close to any new order regardless of which policy is choosing among them, so the auction's more deliberate bid comparison has little idle capacity left to meaningfully reallocate relative to the simpler greedy nearest-agent rule. This is consistent with the observation in Section 7.1 that all three policies perform near-identically under the four mild, individual fault conditions and only diverge once demand pressure (order surge, combined) pushes the fleet toward saturation, at which point the auction's fairness term appears to trade a small amount of average wait time for more even distribution of work, a hypothesis Section 9 also identifies as worth testing directly at higher utilization levels in future work.

9. Limitations

This study is a pilot-scale, fully simulated reference implementation, and its scope should be read accordingly.

- Scale: six agents on a 20 by 14 cell grid is a small reference deployment relative to commercial warehouse fleets, which may run hundreds of vehicles across substantially larger floor plans. The structural findings regarding static zone assignment's load-imbalance vulnerability should generalize qualitatively to larger fleets, since the underlying mechanism (a fixed territory cannot borrow idle capacity from a neighboring territory) does not depend on fleet size, but the specific numeric magnitudes reported in Section 7 are specific to this configuration and should not be read as universal constants.
- Simulation, not hardware: this study models AGV motion, battery depletion, and fault occurrence in software; it does not account for real-world effects such as sensor noise, wheel slip, network latency between a physical robot and a central dispatcher, or the physical time cost of a robot actually grasping and placing an inventory item, which is abstracted here as a fixed two-tick dwell time.
- Bidding uses a distance proxy: the market-auction policy's bid computes Manhattan distance rather than the true Cooperative A* path cost for every candidate agent on every pending order, a deliberate tradeoff made to keep a 144-trial experiment computationally tractable. Manhattan distance is a reasonable lower-bound proxy for actual travel cost on this grid, but it is not identical to it, particularly when congestion faults are active; a version of the auction that computed true path cost per bid might behave somewhat differently under the congestion condition specifically.
- Fault co-occurrence: the combined condition schedules one instance of each of the four fault types staggered so that no two are ever simultaneously active, isolating the cumulative effect of facing all four fault types across one trial. It does not test genuinely simultaneous multi-fault scenarios (for example, an agent breakdown occurring during an active order surge), which is a materially different and likely more severe stress condition left to future work.
- Heartbeat-timeout confound: as discussed in Section 8, the study as designed cannot separate the market-auction policy's fault-recovery advantage into a component attributable to its bidding logic versus a component attributable simply to its shorter heartbeat-timeout constant, because no trial was run with a nearest-idle policy using a shortened timeout. This is the single most direct follow-up experiment this paper identifies.
- No guarantee of acceptance or of applicability implied: this paper presents a reference implementation and an honest, statistically tested comparison at pilot scale; it does not claim that any of the three policies studied here is production-ready, nor that the specific numeric results reported would reproduce unchanged in a different warehouse layout, fleet size, or demand pattern.

10. Conclusion

This paper presented a real, reproducible, 144-trial simulation study comparing three deterministic, fully auditable orchestration policies for a warehouse AGV fleet under four real injected fault conditions plus a combined-stress condition. The results support two specific, evidence-backed conclusions rather than a single sweeping claim. First, a fixed, non-adaptive zone-assignment dispatch scheme carries a real and, under demand stress, statistically severe cost relative to any form of dynamic dispatch, a finding directly actionable for warehouse operators still running fixed-territory dispatch. Second, the specific proposed multi-signal market-auction orchestrator, despite its additional complexity, does not outperform a much simpler reactive nearest-idle heuristic on throughput or wait time in this experiment, and its one clear advantage, faster recovery from agent breakdown, traces to a single configuration parameter rather than to the sophistication of its bidding mechanism. We report both conclusions, including the one that does not flatter the paper's own proposed method, in the interest of giving practitioners and future researchers an honest, statistically grounded basis for deciding how much orchestration complexity a given warehouse deployment actually needs.

Author Contributions

Sai Kumar Pola: simulation architecture, warehouse and agent modeling, Cooperative A* integration, methodology and system design writing. Pavan Kumar Koneru: orchestration policy design including the market-auction mechanism, fault-taxonomy co-design, discussion of operational implications. Rajesh Kodali: statistical analysis and reporting pipeline, results tables, validation of summary statistics against underlying trial data; contribution scoped to data engineering and analytics consistent with this author's professional background.

References

1. Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100-107.
2. Smith, R. G. (1980). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, C-29(12), 1104-1113.
3. Silver, D. (2005). Cooperative Pathfinding. *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 117-122.
4. Gerkey, B. P., & Mataric, M. J. (2004). A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *The International Journal of Robotics Research*, 23(9), 939-954.
5. Chandra, T. D., & Toueg, S. (1996). Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2), 225-267.
6. Wurman, P. R., D'Andrea, R., & Mountz, M. (2008). Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1), 9-19.
7. Khamis, A., Hussein, A., & Elmogy, A. (2015). Multi-robot Task Allocation: A Review of the State-of-the-Art. In *Cooperative Robots and Sensor Networks 2015 (Studies in Computational Intelligence, Vol. 604)*, 31-51. Springer.
8. Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 219, 40-66.
9. Lackner, T., Hermann, J., Kuhn, C., & Palm, D. (2024). Review of Autonomous Mobile Robots in Intralogistics: State-of-the-Art, Limitations and Research Gaps. *Procedia CIRP*, 130, 930-935.

Appendix A: Simulation Parameters Reference

Category	Parameter	Value
Environment	Grid dimensions	20 x 14 cells
Environment	Shelf blocks	4, each 2 columns x 9 rows
Environment	Pack stations	3
Environment	Charging stations	2
Agent	Fleet size	6 agents
Agent	Starting battery	100 units
Agent	Battery move cost	1.0 unit/cell
Agent	Low battery threshold	25 units
Agent	Critical battery threshold	8 units
Agent	Charge rate	6.0 units/tick
Pathfinding	Algorithm	Cooperative A* (space-time)
Pathfinding	Max search horizon	80 ticks
Orders	Arrival process	Poisson, base rate 0.08/tick
Orders	Pick/drop dwell time	2 ticks each
Faults	Agent breakdown duration	40 ticks
Faults	Congestion duration	60 ticks (6-cell aisle segment)
Faults	Battery drain duration	50 ticks (+3.0 units/move)
Faults	Order surge duration	80 ticks (4x rate multiplier)
Experiment	Trial horizon	1,200 ticks + up to 400-tick drain
Experiment	Seeds per cell	8
Experiment	Total trials	144