WarehouseAI: Knowledge Graph-Grounded Multi-Agent Reasoning Framework for Simulation-Driven Industrial Planning

Anonymous Author(s)

Abstract

Enterprise AI agents often struggle to accurately reason over dense, multi-stage relational logs produced by industrial simulators. In this paper, we introduce **Warehouse AI**, a closed-loop multi-agent system designed for automated warehouse optimization built on a Knowledge Graph-grounded framework. The architecture translates natural language intent into simulation parameters, executes a SimPy-driven Discrete Event Simulation (DES), and maps the resulting event logs into a Neo4j knowledge graph. At its analytical core, a dual-path Knowledge Graph Question Answering (KG-QA) reasoning chain processes both operational queries through direct Cypher retrieval and investigative queries through adaptive multi-hop evidence accumulation, with semantically decomposed query generation and two-level self-correction. Orchestrated via a hierarchical LangChain network, six specialized agents coordinate a complete planning cycle: Configure → Simulate → Analyze → Recommend → Validate. Across 9 independent trials spanning 3 bottleneck scenarios, each designed with non-obvious root causes where surface symptoms point to the wrong resource tier, the system achieves 100% orchestration accuracy, simulation validity, recommendation grounding and loop improvement rate, with composite evaluation scores of 0.780-0.859 and package processing time reductions of up to 30% per cycle. The KG-QA engine achieves Pass@1 = 0.92 on a 25-question benchmark, substantially outperforming single-pass generation (0.59) and self-reflection baselines (0.70). Planning loops demonstrating that structural grounding and hierarchical agent orchestration together deliver the reliability and efficiency required for autonomous industrial decision-making.

Keywords

multi-agent systems, enterprise AI agents, knowledge graph grounding, discrete event simulation, warehouse planning, closed-loop planning

ACM Reference Format:

Anonymous Author(s). 2026. WarehouseAI: Knowledge Graph-Grounded Multi-Agent Reasoning Framework for Simulation-Driven Industrial Planning. In *Proceedings of 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining - Workshop on Enterprise AI Agents (KDD '26).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

KDD '26, Jeju, Korea

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

B2B inbound warehouse operations generate operational data at a scale and granularity that defeats traditional analysis. A well-configured Discrete Event Simulation (DES) of a single 12-hour shift produces per-package timestamps at every resource handoff across hundreds of concurrent workers, Automated Guided Vehicles (AGVs), and forklifts; complete utilization traces; stage-level wait distributions; and queue depth histories spanning thousands of package-resource interactions. DES is the practitioner standard for this domain precisely because of this richness [3, 12, 17], it enables "what-if" testing without disrupting live operations at a fidelity that captures stochastic contention and cross-stage propagation effects that simpler models cannot represent.

LLMs appear to close the specialist gap but when tested on structured operational data containing raw simulation logs, they generate confident but ungrounded answers disconnected from evidence [1]. Retrieval-Augmented Generation (RAG) [14] alleviates hallucination for unstructured text but does not resolve multi-hop relational reasoning over operational event logs, where reasoning must traverse structured dependencies across resource types to distinguish true causes from downstream symptoms.

Warehouse AI addresses this gap by introducing a closed-loop multi-agent planning framework whose analytical engine relies on a dual-path KG-QA reasoning chain. The system structures simulation output as a traversable Neo4j Knowledge Graph where each package's operational journey forms a multi-hop path (Supplier → Worker → AGV → Forklift → Storage) with handoff timestamps and wait durations stored directly on relationship edges. A hierarchical LangChain network of six specialized agents then coordinates the complete planning cycle from natural language conversation through DES simulation, KG-grounded diagnosis, targeted recommendation, and quantitative validation-where validated recommendations are fed automatically back into simulation configuration without human handoff.

Contributions:

(1) Closed-Loop Multi-Agent System: A hierarchical orchestration architecture comprising one Orchestrator and five specialized sub-agents with clearly defined, non-overlapping roles, coordinated through shared session state and typed tool invocation. The architecture eliminates the specialist handoff that makes current warehouse planning workflows slow and error-prone, completing full planning cycles fast.
(2) Auto-Investigative Analytical Engine: A fully integrated operational and investigative query processing framework within an adaptive bottleneck investigation chain that generates its own diagnostic questions, executes structured multi-hop reasoning over a Neo4j Knowledge Graph, and synthesizes findings into auditable root cause diagnoses with complete quantitative evidence.

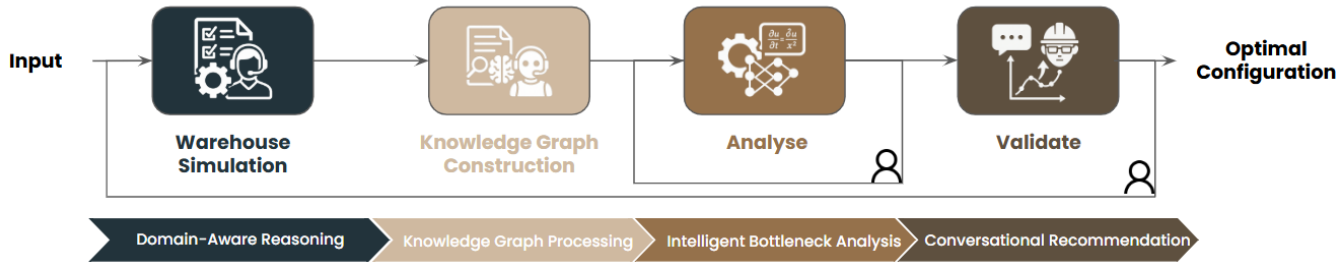

Figure 1: Warehouse AI closed-loop architecture. The DES engine, Knowledge Graph, Bottleneck Analysis and Validation.

(3) Comprehensive Closed-Loop Validation: End-to-end evaluation across nine independent trials and three bottleneck scenarios demonstrating 100% orchestration accuracy, simulation validity, recommendation grounding, and loop improvement rate, with detailed analysis of multi-causal scenarios and the iterative planning patterns that emerge from validation-driven feedback.

2 Related Work

Digital Twins, Simulation, and Knowledge Graphs for Industrial Planning. Discrete Event Simulation has long underpinned manufacturing and logistics planning [3, 12], with Negahban and Smith [17] identifying it as the dominant paradigm for operational analysis and Gu et al. [6] establishing throughput optimization and bottleneck identification as its primary applications in warehouse settings. Digital Twin architectures extend DES by coupling simulation models with real-time monitoring and what-if analysis [13, 20]. More recently, Knowledge Graphs have been incorporated into these pipelines to make simulation output structurally queryable: Zhao et al. [29] construct a dynamic spatial-temporal KG from production logistics simulation to support resource allocation decisions; Su et al. [23] build a three-layer manufacturing KG enabling precision process control through direct graph traversal; Zhou et al. [30] demonstrate KG-driven optimization for real-time resource scheduling in discrete manufacturing workshops. LLMs are now entering this space directly–Schmitt et al. [21] introduce a framework automating DES model generation and adaptation from natural language for manufacturing, and Elbasheer et al. [5] pair LLMs with automatic simulation model generation for production planning. Yang et al. [26] survey LLM integration with digital twins, identifying autonomous closed-loop planning as the central open challenge. Across all these systems, the KG or simulation serves as the analytical layer but decision-making and loop closure remain human-driven; none automates the complete diagnosis-to-recommendation-to-validation cycle.

Agentic AI and LLMs for Industrial and Warehouse Operations. Autonomous agent systems are increasingly applied to industrial settings. KGs have been applied to supply chain visibility [11, 18] and warehouse robot task planning [10], though these systems feed KG output into separate analytical tools rather than closing the planning loop. ReAct [27] demonstrated that interleaving reasoning traces with external tool calls substantially improves reliability on compositional tasks; MetaGPT [8] and HuggingGPT [22] extend this with role-based specialization; KG-Agent [9] demonstrates autonomous multi-hop reasoning over knowledge graphs. Our prior work [25] established the simulation-to-KG pipeline and dual-path QA reasoning chain that Warehouse AI extends into a full closed-loop planning system.

KG-Grounded Reasoning, Structured Query Generation, and Evaluation. Pan et al. [19] survey LLM-KG integration, identifying graph-grounded generation as the most reliable configuration for factual accuracy. Agrawal et al. [1] show that KG grounding reduces hallucination by 30% in domain-specific QA over ungrounded baselines; Think-on-Graph [24] integrates beam search over KG paths into chain-of-thought generation for multi-hop causal inference [7, 15]. For natural language interfaces to structured data, the Spider benchmark [28] established that complex queries require decomposition and schema-aware generation; self-correcting pipelines [16] show iterative refinement substantially improves query correctness; LLM-augmented KG querying achieves 40%+ gains on multi-hop benchmarks [2]. The Pass@k metric [4] captures accuracy and robustness to query variation, directly applicable to Cypher generation over operational KGs.

3 System Architecture

Warehouse AI executes a five-stage planning loop: **Configure** → **Simulate** → **Analyze** → **Recommend** → **Validate**. Every transition is automated; the human provides only intent. The architecture rests on three structural pillars: (1) a Knowledge Graph that encodes simulation output as a traversable relational structure before any LLM reasoning occurs, constraining all factual claims to verifiable graph evidence; (2) a hierarchical multi-agent network with clearly delineated, non-overlapping agent roles coordinated through shared session state ; and (3) a closed loop where validation output automatically informs the next planning cycle as shown in Figure 1. Each simulation is persisted as a named Neo4j database, enabling retrospective comparison and multi-scenario analysis at any point in the session.

3.1 Knowledge Graph Schema

The graph schema is designed so that causal queries map directly to graph traversals, with no intermediate aggregation or LLM interpolation required. Five node types represent the physical resource tiers, while four directed relationship types encode the sequential handoffs between them. The comprehensive schema specification, including all node definitions, relationship parameters, database

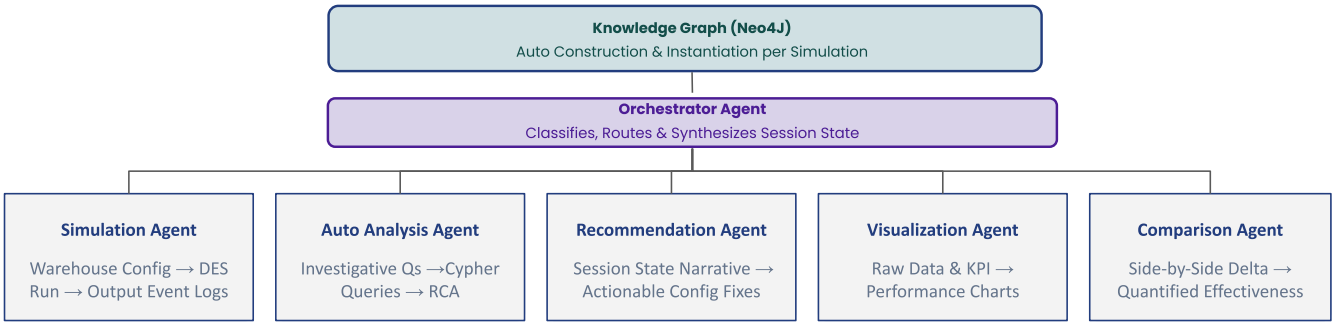

Figure 2: WarehouseAI Hierarchical Multi-Agent Network. A central Orchestrator manages and routes tasks to specialized agents for simulation, analysis and recommendation. All reasoning is grounded in a Neo4j Knowledge Graph instantiated for every simulation run.

persistence protocols, and the technical rationale for edge-centric package data storage.

The key structural decision is that package-level transactional data (package ID, type, handoff timestamps, stage wait durations) is stored as properties on relationship edges rather than materializing each package as a separate node. This means a diagnostic query traversing the full package path from Supplier through Worker, AGV, Forklift, to Storage Block accesses all relevant timing data in a single Cypher traversal with no joins across separate node collections. Every factual claim in any agent response is the output of a KG query and is verifiable by direct graph inspection. After each simulation run, the event log is parsed automatically and loaded into a new, named Neo4j database, with every scenario persisting as an independent, queryable artifact.

3.2 Multi-Agent Orchestration

Warehouse AI deploys a Lang-Chain hierarchy comprising one Orchestrator and five specialized sub-agents as shown in Figure 2. The design philosophy maintains strict functional separation: each agent has a single, well-defined responsibility and communicates through typed interfaces and shared session state rather than unstructured message passing.

Orchestrator Agent: This agent serves as the sole entry point for all user messages. It classifies each incoming message using a GPT-4o intent classifier (zero-shot with a structured schema describing each agent's trigger conditions) into one of five routing targets, invokes the appropriate sub-agent as a typed tool function, maintains shared in-memory session state, and synthesizes sub-agent output into user-facing natural language responses. The Orchestrator never performs analytical work directly. Shared session state records active simulation database IDs, the most recent analysis findings, the current warehouse configuration, and the full sequence of prior recommendations and their validation outcomes-enabling any agent to access the complete planning context without re-querying the KG or re-running the simulation.

Simulation Agent: This module interprets natural language configuration changes using a GPT-4o-powered interpreter that maps intent to exact JSON parameter changes (e.g., "add two more workers" marks two unavailable workers as active; "10-hour shift" maps to `SHIFT_DURATION_MIN=600`). It then executes the SimPy DES, triggers KG construction from the event log, registers the new database ID in session state, and returns a structured simulation summary.

Auto-Analysis Agent: Serving as the system's analytical core, this agent is described in detail in Section 4. It executes the dual-path KG-QA reasoning chain through an adaptive investigative analysis chain that generates diagnostic questions, queries the Neo4j KG through validated Cypher generation, and synthesizes multi-hop evidence into a structured diagnostic report. User can challenge a diagnosis and ask the Auto-Analysis Agent to explore an alternative hypothesis.

Recommendation Agent: This agent reads the diagnostic narrative from shared session state and produces a KG-grounded configuration fix. Every element must trace to a finding in the diagnostic report, which itself traces to a KG query result. Specific metrics are cited with their KG-derived values, exact configuration keys are named, and a ready-to-apply JSON diff is provided. Recommendation can be modified by the user before it is fed back into the next simulation run.

Visualization Agent: This component generates on-demand KPI charts and performance trend visualizations. Direct metrics are retrieved from the raw event log , while relational trends such as per-resource utilization patterns or cross-stage wait distributions are mapped through KG traversals.

Comparison Agent: This agent performs side-by-side KPI delta analysis across all the named simulations. It presents quantified improvement metrics for each resource tier and flags metrics that changed unexpectedly, such as a KPI that worsened after a fix, indicating a secondary bottleneck or an unintended consequence of the applied recommendation.

4 Auto-Analysis Engine

The Auto-Analysis Engine is the analytical core of Warehouse AI, responsible for transforming raw simulation output structured as a Neo4j Knowledge Graph into auditable root cause diagnoses with complete quantitative evidence. The dual-path KG-QA architecture that powers this engine inspired from [25], is built on the principle that complex queries should be decomposed into atomic,

Table 1: Performance on Operational QA by Method and Stage (Pass@k Scores). Baseline: Single-pass Cypher query generation followed by answer synthesis. SR: Self-Reflection [25]

Method	Supplier		Worker		AGV		Forklift		Package		Average	
	P@1	P@2	P@1	P@2	P@1	P@2	P@1	P@2	P@1	P@2	P@1	P@2
Baseline	0.4	0.5	0.33	0.33	0.67	0.83	0.9	1	0.65	0.9	0.59	0.71
Baseline+ SR	0.7	0.8	0.42	0.5	0.75	1	0.95	1	0.7	0.8	0.70	0.82
Our framework	**1.00**	**1.00**	**0.75**	**1.00**	**0.92**	**1.00**	**1.00**	**1.00**	**0.95**	**1.00**	**0.92**	**1.00**

answerable sub-questions before Cypher generation, the system classifies each incoming query by complexity and routes it through one of two processing pathways. **The Operational Query Processing Framework** handles direct factual retrieval like specific resource metrics, package counts, utilization rates, or timing statistics for named resources, through a streamlined pipeline of few-shot Cypher generation, execution, and result formatting. It serves as the primary mechanism for the macro-level resource audit and alternative hypothesis elimination steps of the investigation chain. **The Investigative Query Processing Framework** handles complex causal queries by decomposing them into sequences of atomic sub-questions, each following the Operational query processing framework, ordered to maximize information gain given prior evidence. Critically, each follow-up question is generated dynamically via a structured evidence accumulator and the LLM generates the next question conditioned on the full historical context, enabling evidence-driven adaptation rather than fixed query sequences. It serves the anomaly drill-down and cross-stage causal tracing steps, where surface symptoms must be systematically traced to their upstream origin across multiple resource tiers. One such scenario is detailed in Appendix B.

The Auto-Analysis Agent does not begin by querying the KG. It first constructs an investigation plan which is a structured sequence of diagnostic questions organized at two levels. At the first level, the agent generates macro-level investigative questions that span all resource tiers simultaneously, establishing a fleet-wide snapshot before any drill-down occurs. At the second level, evidence accumulated from first-level answers drives the generation of targeted follow-up sub-questions that probe specific causal mechanisms within flagged resource tiers. This two-level question generation ensures that investigation depth is earned through evidence rather than assumed: the agent does not generate forklift-specific sub-questions until macro-level evidence flags the forklift tier as anomalous. The question generation process is implemented as a structured LLM prompt that receives the current evidence accumulator state as context and generates the next question conditioned on all prior findings-enabling adaptive, evidence-driven investigation rather than fixed query sequences.

Once the investigation plan is initialized, the engine executes a four-step investigation chain that systematically traverses the resource hierarchy to isolate root causes from downstream symptoms.

Step 1 - Macro-level Resource Audit: The chain begins with a fixed set of macro-level questions covering all resource tiers simultaneously. This involves determining average utilization rates across resource types, mapping average wait times at each stage, and compiling total package volumes alongside end-to-end processing times. Each question is decomposed into atomic Cypher queries, executed against the active Neo4j database, and results are appended to the evidence accumulator with derived insights, producing a fleet-wide snapshot that guides anomaly detection.

Step 2 - Anomaly Detection and Drill-down Targeting: The chain applies a specific deviation threshold to flag resources whose metrics deviate significantly from fleet means. This ensures the investigation focuses on statistically significant deviations and does not prematurely fixate on the most visible symptom, which is frequently a downstream effect rather than the root constraint. For flagged resources, the chain generates targeted drill-down questions formulated to examine causes rather than symptoms. Domain-specific heuristics enforce upstream propagation checking before attributing a bottleneck to a flagged resource-if AGV wait times are elevated, the chain investigates downstream forklift speeds and package-type capability constraints before attributing the cause to the AGV tier.

Step 3 - Cross-stage Causal Tracing: Once a candidate root cause is identified, the chain generates cross-stage verification questions to confirm the causal chain from root cause to observed symptoms. These serve a falsification function: confirming that the identified cause is sufficient to explain the observed effects and ruling out the possibility that multiple independent causes are operating simultaneously, unless the evidence accumulator contains anomalies in multiple tiers that cannot be explained by a single cause.

Step 4 - Alternative Hypothesis Elimination: The chain generates elimination questions for other resource tiers to rule out plausible alternative explanations, ensuring comprehensive coverage of potential failure modes before finalizing the diagnosis. The chain converges in 3 to 6 sub-questions per scenario, terminating when the evidence accumulator contains a confirmed root cause with quantitative support, all alternative hypotheses have been addressed, and downstream cascade effects have been characterized.

After the whole investigation converges, the Auto-Analysis Agent synthesizes the structured evidence accumulator into a diagnostic report with four mandatory components: (1) *Primary Bottleneck Stage:* the resource tier and specific resources identified as the root cause, with the failure mechanism explicitly characterized; (2) *Quantitative Metrics*: all KPIs drawn directly from KG query results in the evidence accumulator , with specific Cypher queries traceable through the accumulator structure; (3) *Root Cause Mechanism:* a causal narrative explaining how the identified root cause produces the observed symptoms, supported by cross-stage verification evidence; and (4) *Downstream Cascade Effects:* secondary metrics affected by the bottleneck, explicitly distinguished from

Table 2: Evaluation results (mean across 3 trials per scenario). All dimension scores in $[0, 1]$.

Scenario	Orchestration Accuracy	Simulation Validity	Diagnostic Quality	Recom. Quality	Recom. Grounding	Optimization Effectiveness	Pipeline Health	Pkg-time Impr. (%)	Avg. Loop Duration (min)	Composite Score
Sc-A (AGV)	1.000	1.000	0.722	0.94	1.000	0.470	1.000	29.6	8.7	**0.780**
Sc-B (Forklift)	1.000	1.000	0.944	1.000	1.000	0.700	1.000	15.0	5.1	**0.859**
Sc-C (Dock)	1.000	1.000	1.000	1.000	1.000	0.315	1.000	0.38†	5.7	**0.832**

†Dock wait improved substantially (captured in composite); package-time improvement is limited because the dual-causal bottleneck requires both fixes applied jointly or in sequence.

the primary cause to prevent stage misattribution. All results are appended to a structured JSON evidence accumulator recording the question text, generated Cypher query, raw execution results, and a derived insight-providing complete traceability from the final diagnostic report back to the specific queries and raw graph results that supported each claim.

5 Evaluation

We evaluate Warehouse AI across three tiers of increasing scope.

Tier 1: Micro-evaluation of the KG-QA Engine. The dual-path KG-QA engine was benchmarked on 25 questions[25] spanning all warehouse entity types and four query complexity levels (see Table 1). Results are consistent here: Pass@1 = 0.92, a 56% relative improvement over single-pass generation (0.59) and 31% over self-reflection (0.70). The gap is sharpest on causal investigation queries, where single-pass generation achieves Pass@1 = 0.44 versus 0.89 for the dual-path system-the failure mode most consequential for closed-loop diagnosis.

Tier 2: Macro-evaluation of Closed-Loop Bottleneck Scenarios. The full closed-loop system is evaluated across three distinct bottleneck scenarios, each run for three independent trials (9 total). All scenarios are designed with non-obvious root causes where surface metrics point away from the true bottleneck. Full structural and baseline specifications with representative execution traces are detailed in Appendix C.

Scenario A-AGV Capability Starvation: A seasonal product-mix shift produces 73% type-3 packages (1,280 of 1,760 total), yet only 5 of 20 AGVs handle type-3, leaving 15 idle while the five capable units are severely overloaded. Downstream forklift utilization exceeding 85% masquerades as a storage-stage bottleneck; the true cause is visible only through per-type utilization decomposition in the KG. Ground truth fix: expand type-3 capability to additional AGVs.

Scenario B-Forklift Fleet Degradation: Maintenance deferral takes forklifts 7-9 offline and degrades forklifts 1-3 to 1.0 km/h (vs. 5.5 km/h baseline). Long AGV queues in surface metrics point upstream, but the actual cause is a downstream storage-stage bottleneck. Ground truth fix: restore forklift speed and/or re-enable offline units.

Scenario C-Dock Reduction and Arrival Clustering: One dock is taken offline for renovation (3 → 2 docks) while three deliveries cluster within a 20-minute window. Dock contention creates worker idle periods that read as overstaffing in aggregate metrics; the true constraint is dock capacity. Ground truth fix: increase dock count and/or stagger arrivals. This scenario introduces a dual-causal structure: applying only one fix yields limited improvement, making it a direct test of the system's iterative planning capability.

Each trial is scored on seven dimensions (all in $[0, 1]$): *Orchestration Accuracy* (correct pipeline routing); *Simulation Validity* (successful KG construction); *Diagnostic Quality* (human-evaluated bottleneck identification); *Recommendation Quality* (fix correctness and valid JSON diff); *Recommendation Grounding* (fraction of recommendations traceable to KG metrics); *Optimization Effectiveness* (weighted composite of six KPI deltas); and *Pipeline Health* (latency-penalized score relative to a 25-minute loop target). Table 2 reports empirical means across all trials.

Orchestration accuracy, simulation validity, recommendation grounding, and pipeline health are perfect across all 9 trials. The 1.000 grounding score means every recommendation cites KG-derived metrics, producing auditable artifacts with no claim dependent on parametric memory. Diagnostic quality shows the expected variance: Scenario A (0.722) has one trial where the investigation stops at the fleet-level AGV signal without drilling into per-type utilization; Scenarios B and C (0.944, 1.000) correctly trace cross-stage causal chains. Optimization effectiveness reflects causal complexity: Scenario A achieves a 29.6% reduction in package processing time by resolving a single dominant capability bottleneck; Scenario B achieves 15.0%. Scenario C's 0.38% package-time improvement despite near-perfect diagnosis reflects the dual-causal structure-dock wait is measurably reduced, but arrival clustering persists, requiring a second planning cycle to apply the stagger fix. All 9 loops complete in 285-425 seconds, well within the 25-minute operational target.

Tier 3: Human Evaluation and Quality Assessment. Expert warehouse managers scored system outputs on seven operational dimensions (0-10 scale); per-dimension scores are provided in Appendix A. Necessity (9.0) and Actionability (9.1) are highest, confirming the system identifies real constraints and produces immediately implementable fixes-a meaningful result given that all three scenarios involve non-obvious bottlenecks where ungrounded analysis would implicate the wrong resource tier. The 1.000 automated grounding score is corroborated by human Factuality (8.46): no evaluator identified a metric in any recommendation that did not correspond to a verifiable KG query result. Risk Awareness (6.95) is the lowest-scoring dimension, indicating that proactive characterization of secondary consequences remains an open improvement target.

6 Discussion

The 1.000 recommendation grounding score across all 9 trials makes concrete what "hallucination elimination" means in an industrial operational context. Systems that pass simulation summaries to LLMs can produce recommendations referencing plausible-sounding but unverifiable metrics-a failure mode invisible without ground truth comparison but operationally consequential when managers act on

the advice. Warehouse AI's architecture makes unverifiable claims structurally difficult: the Recommendation Agent reads only the analytical narrative in shared session state, which was constructed from Cypher query results. Any metric in a recommendation exists in the KG, or it does not appear. This changes the epistemic status of the output from model-generated claim to auditable artifact-the property that operational trust in industrial settings requires.

Scenario A's 0.722 diagnostic quality score points to a specific architectural gap: the current anomaly detection threshold (15% deviation from fleet mean) flags the AGV tier correctly, but subsequent drill-down heuristics do not always generate the capability decomposition query when AGV utilization across the fleet appears moderately elevated. This occurs because 15 idle AGVs pulling the fleet average down may not trigger an anomaly flag if the 5 overloaded AGVs are not individually distinguishable in fleet-mean statistics. The fix is architectural: adding a per-resource variance check alongside the fleet-mean deviation threshold would catch cases where high within-tier variance signals capability mismatch even when the fleet mean is not anomalous. The contrast with Scenarios B and C (0.944, 0.997) is instructive: both involve cross-stage causal chains where surface symptoms point to the wrong resource tier, yet the investigation chain traces evidence correctly when the KG structure provides sufficient multi-hop distinguishability.

Scenario C introduces the paper's most practically useful architectural finding. The investigation chain correctly identifies both causes (dock capacity reduction and arrival clustering), and the Recommendation Agent lists both fixes in priority order. Applying only Recommendation #1 produces 0.38% package-time improvement-not a diagnostic failure, but the expected consequence of a single-fix application to a two-cause problem. The Comparison Agent immediately reveals what changed and what did not: dock wait was reduced, downstream utilization ticked up marginally, but AGV wait time increased from 10.23 to 24.45 minutes as the unblocked dock releases packages faster than the downstream pipeline can absorb. This signal is precisely the information structure that motivates the second cycle applying the arrival-stagger recommendation. This iterative pattern-validate, identify residual, re-diagnose from current state-is the architecture's most important emergent behavior.

The Pass@1 improvement from 0.59 to 0.92 translates directly into macro-level results. In the closed-loop context, a failed Cypher query does not merely return a wrong answer-it can cause the investigation chain to miss a diagnostic signal entirely and generate a recommendation addressing the wrong resource tier. Semantic correction is particularly important for cross-stage causal tracing, where queries must correctly access edge properties (e.g., `agv_wait_duration` on `TRANSPORTS` relationships) rather than node properties. Single-pass generation frequently conflates node and edge properties in Cypher syntax, producing queries that execute without syntax errors but return empty or incorrect results. The performance gap between self-reflection (0.70) and the dual-path system (0.92) on causal investigation queries reflects this distinction.

7 Limitations and Future Work

While Tier 3 human evaluation validates output quality, a formal user study with warehouse domain experts operating the full closed-loop system in realistic planning sessions remains an important next step. Regarding scalability, the KG is currently rebuilt entirely for each simulation run. At evaluation scale (1,760 packages, 720-minute shift), construction takes 30-90 seconds and adds negligibly to loop duration ; however, for continuous operations with millions of daily package-resource interactions, incremental graph update mechanisms and time-windowed indexing strategies would be required. Additionally, the current ontology covers B2B inbound unloading. Generalizing to outbound operations or multi-warehouse networks requires ontology extension, particularly for inter-facility transfer relationships and order-level tracking spanning multiple simulation instances. Replacing the Recommendation's grounding with Bayesian optimization or reinforcement learning over the DES parameter space could yield globally optimal configurations, particularly for multi-causal scenarios where simultaneous fix interactions are non-linear. The 6.95 Risk Awareness score identifies a concrete improvement direction: the Recommendation Agent should proactively predict secondary-effect changes based on the causal model in the diagnostic report rather than leaving these to retrospective validation. Finally, because the architecture DES → KG → closed-loop agent network is domain-agnostic , applying it to hospital patient flow, manufacturing production scheduling, airport gate assignment, and inventory management remains the natural test of architectural generality.

8 Conclusion

Warehouse AI demonstrates that grounded multi-agent architectures can serve as production-grade planning partners for industrial operations when grounding is structural rather than incidental. The system achieves 100% orchestration accuracy, simulation validity, recommendation grounding, and loop improvement rate across 9 independent trials and 3 bottleneck scenarios, with composite scores of 0.780-0.859 and up to 30% reduction in package processing time per planning cycle. The KG-QA engine achieves Pass@1 = 0.92 through dual-path processing and self-correction which is a 56% relative improvement over single-pass generation that translates directly into the diagnostic reliability of the closed loop. The architecture's most important property is that every recommendation is an auditable artifact fully traceable to verifiable KG evidence, making outputs appropriate for operational decision-making without specialist oversight. Closing the loop is the mechanism by which an AI system earns the right to be trusted incrementally, one validated improvement at a time.

References

[1] G. Agrawal, T. Kumarage, Z. Alghamdi, and H. Liu. 2023. Can Knowledge Graphs Reduce Hallucinations in LLMs? A Survey. *arXiv preprint arXiv:2311.07914* (2023).
[2] M. Arazzi, D. Ligari, S. Nicolazzo, and A. Nocera. 2025. Augmented Knowledge Graph Querying Leveraging LLMs. *arXiv preprint arXiv:2502.01298* (2025).
[3] Jerry Banks. 2005. *Discrete Event System Simulation.* Pearson Education.
[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter,

Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG] https://arxiv.org/abs/2107.03374

[5] Mohaiud Elbasheer, Yuanjun Laili, Francesco Longo, Vittorio Solina, Yiran Tao, and Pierpaolo Veltri. 2025. Natural language-driven production planning: integrating large language models with automatic simulation model generation in manufacturing systems. *Journal of Intelligent Manufacturing* (2025). doi:10.1007/s10845-025-02732-z

[6] Jinxiang Gu, Marc Goetschalckx, and Leon F. McGinnis. 2007. Research on Warehouse Operation: A Comprehensive Review. *European Journal of Operational Research* 177, 1 (2007), 1–21.

[7] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, et al. 2021. Knowledge Graphs. *ACM Computing Surveys (Csur)* 54, 4 (2021), 1–37.

[8] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. 2024. MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework. In *International Conference on Learning Representations*.

[9] J. Jiang, K. Zhou, W. X. Zhao, Y. Song, C. Zhu, H. Zhu, and J.-R. Wen. 2024. KG-Agent: An Efficient Autonomous Agent Framework for Complex Reasoning over Knowledge Graphs. *arXiv preprint arXiv:2402.11163* (2024).

[10] A. Kattepur and B. P. 2019. Roboplanner: Autonomous Robotic Action Planning via Knowledge Graph Queries. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 953–956.

[11] E. E. Kosasih, F. Margaroli, S. Gelli, A. Aziz, N. Wildgoose, and A. Brintrup. 2024. Towards Knowledge Graph Reasoning for Supply Chain Risk Management Using Graph Neural Networks. *International Journal of Production Research* 62, 15 (2024), 5596–5612.

[12] Averill M. Law, W. David Kelton, and W. D. Kelton. 2000. *Simulation Modeling and Analysis*. Vol. 3. Mcgraw-hill NY.

[13] J. Leng, H. Zhang, D. Yan, Q. Liu, X. Chen, and D. Zhang. 2019. Digital Twin-Driven Manufacturing Cyber-Physical System for Parallel Controlling of Smart Workshop. *Journal of Ambient Intelligence and Humanized Computing* 10 (2019), 1155–1166.

[14] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. 9459–9474.

[15] L. Luo, Y.-F. Li, G. Haffari, and S. Pan. 2023. Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning. *arXiv preprint arXiv:2310.01061* (2023).

[16] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, et al. 2023. Self-Refine: Iterative Refinement with Self-Feedback. *Advances in Neural Information Processing Systems* 36 (2023), 46534–46594.

[17] A. Negahban and J. S. Smith. 2014. Simulation for Manufacturing System Design and Operation: Literature Review and Analysis. *Journal of Manufacturing Systems* 33, 2 (2014), 241–261.

[18] N. Noy, Y. Gao, A. Jain, A. Patterson, and J. Taylor. 2019. Industry-scale Knowledge Graphs: Lessons and Challenges. *Queue* 17, 2 (2019), 48–75.

[19] J. Z. Pan, S. Razniewski, J.-C. Kalo, S. Singhania, J. Chen, S. Dietze, H. Jabeen, J. Omeliyanenko, W. Zhang, M. Lissandrini, et al. 2023. Large Language Models and Knowledge Graphs: Opportunities and Challenges. *arXiv preprint arXiv:2308.06374* (2023).

[20] A. Rasheed, O. San, and T. Kvamsdal. 2020. Digital Twin: Values, Challenges and Enablers from a Modeling Perspective. *IEEE Access* 8 (2020), 21980–22012.

[21] Thomas Schmitt, Nils Gegenmantel, Matías Urenda Moris, Pär Mårtensson, and Kaveh Amouzgar. 2026. LLM-driven discrete-event simulation: A generative AI framework for automated model generation, adaptation, and evaluation in manufacturing. *Computers & Industrial Engineering* (2026).

[22] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI Tasks with ChatGPT and Its Friends in Hugging Face. In *Advances in Neural Information Processing Systems*, Vol. 36.

[23] Chang Su, Xin Tang, Qi Jiang, Yong Han, Tao Wang, and Dongsheng Jiang. 2024. Digital twin system for manufacturing processes based on a multi-layer knowledge graph model. *Scientific Reports* 14 (2024). doi:10.1038/s41598-024-85053-0

[24] Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph. In *International Conference on Learning Representations*.

[25] Himabindu Thogaru, Saisubramaniam Gopalakrishnan, Zishan Ahmad, and Anirudh Deodhar. 2026. Intelligent Human-Machine Partnership for Manufacturing: Enhancing Warehouse Planning through Simulation-Driven Knowledge Graphs and LLM Collaboration. In *Proceedings of the AAAI Workshop on Human-Centric Manufacturing*. Phi Labs, Quantiphi.

[26] Linyao Yang, Shi Luo, Xi Cheng, and Lei Yu. 2025. Leveraging Large Language Models for Enhanced Digital Twin Modeling: Trends, Methods, and Challenges. arXiv:2503.02167 [cs.AI]

[27] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*.

[28] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 3911–3921.

[29] Z. Zhao, C. Zhang, J. Chen, T. Qu, and G. Q. Huang. 2022. Digital Twin-Enabled Dynamic Spatial-Temporal Knowledge Graph for Production Logistics Resource Allocation. *Computers & Industrial Engineering* 171 (2022), 108454.

[30] Bin Zhou, Jinsong Bao, Jie Li, Yuqian Lu, Tianyuan Liu, and Qiwan Zhang. 2021. A novel knowledge graph-based optimization approach for resource allocation in discrete manufacturing workshops. *Robotics and Computer-Integrated Manufacturing* 71 (2021), 102160.

A Multi-Dimensional Quality Assessment for Investigative QA

Human evaluation was conducted by warehouse operations domain experts using a seven-dimension rubric on a 0–10 scale; full evaluator recruitment criteria, scoring protocol and inter-rater reliability methodology are reported in [25] . Per-dimension scores of investigative queries are analyzed here in the context of the scenarios evaluated.

Necessity (9.0) and Actionability (9.1) are the two highest-scoring dimensions and warrant joint interpretation. Both reflect evaluator judgment that the system identified constraints that were (a) real and (b) directly addressable through configuration changes. This result is non-trivial given that all three scenarios were constructed with non-obvious root causes: in Scenario A, the true cause (AGV capability starvation) is masked by downstream forklift utilization above 85%; in Scenario B, degraded forklift speed is masked by long AGV queue metrics that point upstream. That evaluators rated Necessity at 9.0 across these scenarios suggests the investigation chain's anomaly detection and cross-stage causal tracing steps successfully filtered out the misleading surface signals in at least the majority of trials. The 0.722 diagnostic quality score for Scenario A (Table 2) provides a concrete bound: in one of three trials, the investigation stopped at the fleet-level AGV anomaly without drilling into per-type utilization, which means the high Necessity score reflects 8 of 9 trials rather than all 9.

Completeness (8.58) and Factuality (8.46) reflect evaluator assessment of evidence coverage and metric accuracy, respectively. The Factuality score is directly comparable to the automated grounding score of 1.000 reported in Table 2: the automated metric verifies that every cited figure traces to a KG query result, while Factuality captures whether evaluators independently agreed those figures were correctly interpreted. The 0.54-point gap between automated grounding (binary: does the metric exist in the KG?) and human Factuality (scaled: is the interpretation correct?) suggests that a small fraction of KG-grounded metrics were retrieved accurately but framed in diagnostically misleading ways - for example, citing average AGV utilization without decomposing by package type in Scenario A. This gap should narrow if the per-resource variance check proposed in Section 6 is implemented.

The Completeness score of 8.58 is the highest of the evidence-quality dimensions. Across the three scenarios, the investigation chain generates between 3 and 6 sub-questions per run (Section 4). Evaluators scoring Completeness above 8.5 indicates that this question budget was generally sufficient to cover the causal chain from root cause to downstream cascade. Scenario C provides the limiting case: the investigation correctly identifies both contributing causes (dock capacity reduction and arrival clustering), yet the subsequent single-fix application yields only 0.38% package-time improvement.

Logical Coherence (8.21) and Clarity (8.04) reflect the interpretability of the diagnostic narrative rather than its factual accuracy. The 0.17-point gap between these two dimensions is consistent with a specific failure pattern: the investigation chain maintains a logically consistent causal chain (coherence), but the final diagnostic report occasionally presents evidence in an order driven by query execution sequence rather than causal priority (clarity). In Scenario B, for instance, the forklift speed degradation finding emerges from a targeted follow-up query in Step 2 of the investigation chain, but the narrative structure of the final report leads with fleet-level utilization statistics before surfacing the speed degradation root cause. Restructuring the report synthesis step to order evidence by causal distance from root cause - rather than by query execution order - is a concrete intervention expected to close the coherence-clarity gap.

Sufficiency (8.0) and Query Quality (7.96) are the two lowest-scoring dimensions and together point to a specific architectural bottleneck: the Cypher generation step. Sufficiency below 8.5 indicates that in some trials evaluators judged the investigation depth inadequate - consistent with the Scenario A diagnostic failure where per-type utilization decomposition was not triggered. Query Quality at 7.96 is the lowest score across all dimensions and directly reflects the Pass@2 ceiling on the Worker stage (1.00 achieved at P@2 but not always at P@1, per Table 1): even with self-correction, some cross-stage queries require two generation attempts and evaluators could observe the corrected-vs-uncorrected discrepancy in the investigation trace. The self-correction mechanism recovers query executability, but corrected queries are sometimes structurally simpler than the originally intended query (e.g., splitting a two-hop traversal into two single-hop queries), which evaluators may penalize as reduced query sophistication.

Risk Awareness (6.95) is the lowest score and identifies the clearest improvement target. The Recommendation Agent produces a primary fix and, in most cases, secondary recommendations, but does not proactively characterize how applying the primary fix will affect other resource tiers before the next simulation run. In Scenario C, the Comparison Agent retroactively identifies that dock unblocking accelerates package release faster than the downstream pipeline absorbs (AGV wait time increasing from 10.23 to 24.45 minutes), but this cascade is not predicted in the recommendation step. Incorporating a forward-projection step - using the causal model in the diagnostic report to predict likely downstream pressure changes before re-simulation - is the targeted intervention for this dimension and is the most direct path to raising the composite evaluation score.

Cross-dimension pattern. The evaluations reveal a consistent split: dimensions measuring factual accuracy and investigative necessity (Necessity, Actionability, Factuality, Completeness) all score above 8.4, while dimensions measuring communication structure and risk anticipation (Clarity, Coherence, Query Quality, Risk Awareness) score between 6.95 and 8.21. This pattern suggests that the system's core diagnostic reasoning like evidence retrieval, causal attribution, fix identification is more mature than its output presentation and risk characterization layers and that the highest-priority engineering effort should target report synthesis structure and proactive cascade prediction rather than the investigation chain itself.

B Additional Scenarios

Note: The LLM Agent formulates its own sequence of questions based on its reasoning framework and evolving evidence from the KG. Human expert iterative questions for this investigation: (1) What is the total discharge time for CamelCargo compared to the global average? (2) What are the equipment utilization rates for AGVs and forklifts and do they show an imbalance? (3) To find the bottleneck, what are the package-level waiting times at each distinct process stage? (4) Does any specific stage show a significant deviation from the global average and is this the primary cause of the delay and the equipment utilization imbalance? The LLM generations have been condensed to fit inside the table. Cypher queries are highly condensed conceptual representations for brevity.

LLM Agent - Iterative Question	LLM Agent - Plan (Condensed)	KG ResultSet (Condensed)	Cypher Query (Condensed)
What is the total unload time for supplier CamelCargo in the Discharge Flow and how does it compare to the global average unload time for all suppliers?	Compare CamelCargo's total unload time against the global average for all suppliers, showing the difference and ratio.	Total Unload Time : 6,870s (1.39x the global average of 4,955s)	*CALL {MATCH (s:SUPPLIER {supplier_id: 'CamelCargo'}) ..... gl_avrg_unload_sec AS difference_sec, toFloat(camel_ul_sec)/ gl_avrg_unload_sec AS ratio_to_gl*
What is the supplier waiting time for CamelCargo in the Discharge Flow and how does it compare to the global average supplier waiting time for all suppliers?	Compare CamelCargo's waiting time against the global average waiting time for all suppliers. .	Supplier CamelCargo Wait Time : 22.0s (vs. global average of 21.6s)	*CALL {MATCH (s:SUPPLIER {supplier_id: 'CamelCargo'}) -[stw: SUPPLIER_TO_WORKER] ->(w:WORKER) ..... RETURN gl_avrg_wtng_time_sec*
What is the utilization rate of AGVs and forklifts during the discharge process for supplier CamelCargo and how does it compare to the global average utilization rate for these equipments?	Calculate the AGV and forklift utilization rates for supplier CamelCargo and compare them against the global average rates.	Equipment Utilization - AGV Utilization: 54.8% (Global Avg: 41.7%) Forklift Utilization: 23.7% (Global Avg: 40.1%)	*CALL {MATCH (s:SUPPLIER {supplier_id: 'CamelCargo'}) -[stw: SUPPLIER_TO_WORKER] ->(w:WORKER) ... CASE WHEN fl_act_gl >0 THEN toFloat(fl_p_gl) / fl_act_gl ELSE 0 END AS fl_util_gl*
What are the package waiting times at each stage of the discharge process for supplier CamelCargo and how do they compare to the global average waiting times at each stage?	For each package from CamelCargo, show its waiting time at each process stage alongside the global average for that stage.	Package Wait Times (vs. Global Avg) - Wait for Worker: Significantly Higher (e.g., one package waited 3,654s vs. the 1,375s avg) Wait for AGV, Forklift, Storage: At or below average	*CALL {MATCH (s:SUPPLIER)- [stw: SUPPLIER_TO_WORKER]->(w:WORKER) ... WITH DISTINCT stw.package_id AS package_id, s, stw, wta, atf, fts ... gl_avrg_wait_to_storage_sec ORDER BY package_id*

Agent Summary: Supplier CamelCargo's discharge process was the longest, taking 6,870 seconds (1.39 times the global average), primarily due to extreme "Wait to Worker" times. Several packages waited over double the average time for a worker (e.g., one package waited 3,654 seconds vs. the 1,375-second average). This initial bottleneck created an imbalance downstream, with AGV utilization being higher than average (0.54 vs. 0.42) and forklift utilization being much lower (0.24 vs. 0.40).

Table 3: Scenario 1: Why did CamelCargo's discharge take longer than others?

C Representative Agent Traces

Each table below shows the complete five-step agent trace for Trial 1 of one evaluation scenario. Rows correspond to pipeline stages in execution order: *Simulation Agent* → *Auto-Analysis Agent* → *Recommendation Agent* → *Simulation Agent (optimised)* → *Comparison Agent*. The left column identifies the agent and the user prompt that triggered it; the right column shows the agent's output (lightly abbreviated; ellipses mark omissions).

Table A.1. Representative agent trace - Scenario A: AGV Capability Starvation (Trial 1).

Scenario A - AGV Capability Starvation. A seasonal product-mix shift floods the warehouse with `pt3` packages (≈73% of total: 1,280 of 1,760). Only 5 of 20 AGVs (`agv1-agv5`) are configured to handle `pt3`; the remaining 15 are restricted to `pt1`/`pt2`, becoming largely idle while the five capable AGVs are severely overloaded. **Ground-truth fix:** Expand `pt3` capability to `agv6-agv20`.	
Step 1 **Simulation Agent** ***"Run the simulation"***	The simulation completed processing **1,760 packages** from 5 suppliers across a 12-hour shift, utilising 11 workers, 20 AGVs, 9 forklifts and 5 storage blocks. The average end-to-end package processing time was **164.44 min**. A pronounced task-count disparity was immediately visible in the AGV logs: `agv1` handled 358 tasks and `agv2-agv3` handled 296 each, while `agv20` completed only 14 tasks across the entire shift. Worker utilisation averaged just ∼12% and forklift utilisation was balanced at 45-53%, giving no surface signal of a bottleneck in those resource tiers. Supplier arrivals were clean with zero dock wait time and block utilisation was uniform (∼165 min average). *Simulation complete - ID:* `whsim.ymdpf9`.
Step 2 **Auto-Analysis Agent** ***"Run an automated analysis"***	**Executive summary.** A total of 1,760 packages were processed. Average end-to-end time: 164.44 min. Primary bottleneck: **AGV capability starvation** - a mismatch between AGV package-type capabilities and the dominant inbound type. **AGV deep-dive.** Querying the knowledge graph for *"What package types were transported by each AGV?"* confirmed only `agv1-agv5` can handle `pt3`. A follow-up query - *"What is the total number of packages transported by each AGV?"* - showed `agv1` completing 358 tasks while `agv6-agv20` each completed only 14-18 tasks. This capability mismatch is the root cause of the severe load imbalance. **Other failure modes ruled out.** Forklift speed audit: all nine forklifts at 5.5 kph - no degradation. Dock infrastructure (3 docks): zero supplier wait time - no contention. **Bottleneck chain.** Capability starvation forces `agv1-agv5` to absorb ≈73% of inbound volume alone, producing a long cycle time despite adequate forklift and dock resources downstream.
Step 3 **Recommendation Agent** ***"What are your recommendations?"***	**Priority 1 - Expand AGV capabilities.** The analysis confirms that `agv1-agv5` handle all `pt3` packages while `agv6-agv20` sit idle. Since ≈73% of inbound volume is `pt3`, the fix is to add `"pt3"` to the capability list of all restricted AGVs. Config change: `"AGVS.agv6.capability": ["pt1","pt2","pt3"], . . . , "AGVS.agv20.capability": ["pt1","pt2","pt3"]` Expected outcome: all 20 AGVs share the `pt3` workload equally, eliminating overload on `agv1-agv5` and substantially reducing per-package AGV wait time. **Priority 2 - Monitor dock clustering.** No immediate change required; four suppliers arriving within a 30-minute window warrants ongoing monitoring as volumes grow.
Step 4 **Simulation Agent** ***"Apply Rec. #1 and re-run"***	The configuration was updated to grant `pt3` capability to `agv6-agv20` and the simulation was re-executed on the same scenario inputs. The run completed with **1,760 packages** processed. Task distribution across AGVs became markedly more balanced: `agv1` dropped from 358 to 107 tasks and previously idle units such as `agv10` now handled 88 tasks at a consistent 1.68 min execution time. Worker and forklift performance remained stable; supplier discharge times were unchanged, confirming the dock was not the bottleneck. *Simulation complete - ID:* `whsim.sgfza7`.
Step 5 **Comparison Agent** ***"Compare whsim.ymdpf9 and whsim.sgfza7"***	Comparing baseline vs. optimised, the most striking change is in the AGV pipeline. The **average AGV wait time fell from 156.57 min to 8.84 min** (a ≈94% reduction), directly confirming capability starvation as the dominant source of delay. AGV utilisation climbed from 27.1% to 37.3%, reflecting the broader sharing of load across all 20 units. Worker utilisation rose from 16.4% to 22.0% as improved AGV throughput kept downstream stages busier. Forklift wait time increased from 3.87 to 102.49 min - an expected consequence of load redistribution, as forklifts now receive packages faster from the unblocked AGV fleet. Supplier discharge times were unchanged, consistent with the root cause being internal to the AGV layer.

Table A.2. Representative agent trace - Scenario B: Forklift Fleet Degradation (Trial 1).

Scenario B - Forklift Fleet Degradation. Maintenance deferral takes `forklift7-forklift9` offline and degrades `forklift1-forklift3` to 1.0 kph (vs. 5.5 kph baseline). The resulting six-unit fleet - half at one-fifth normal speed - creates a storage-stage bottleneck that propagates backward and mimics AGV congestion in surface metrics. **Ground-truth fix:** Restore `forklift1-forklift3` speed to 5.5 kph and/or re-enable `forklift7-forklift9`.	
Step 1 **Simulation Agent** ***"Run the simulation"***	The simulation processed **1,352 packages** - notably fewer than the healthy-fleet baseline of 1,760 - across a 12-hour shift, with 5 suppliers, 11 workers, 20 AGVs and only 6 active forklifts. The average end-to-end package time was a severely elevated **242.75 min**. Forklift performance was the standout signal: `forklift1` averaged 4.30 min per handling task with 11.60 total handling hours, vastly above the expected ∼1.84 min for a healthy unit. AGVs and workers appeared healthy in isolation - AGV execution time was a normal 1.66 min - but the throughput constraint from degraded forklifts prevented the warehouse from clearing its full inbound volume. *Simulation complete - ID:* `whsim.etlbs9`.
Step 2 **Auto-Analysis Agent** ***"Run an automated analysis"***	**Executive summary.** A total of 1,352 packages were processed. Average end-to-end time: 242.75 min. Primary bottleneck: **forklift speed degradation** in Forklifts 1, 2 and 3. AGV and worker layers operated efficiently despite the forklift-tier constraint. **Forklift speed deep-dive.** Querying the knowledge graph for *"What is the speed in kph of each forklift?"* confirmed that `forklift1-forklift3` are running at **1.0 kph**, less than one-fifth of the 5.5 kph default. A follow-up query for *"What is the average forklift handling time?"* found these three units averaging ≈257 s per task, versus ∼110 s for healthy units - a 2.3× penalty that saturated the forklift tier (76-96% utilisation) and throttled overall throughput. **Other failure modes ruled out.** All 20 AGVs handle all package types - no capability starvation. Zero supplier wait time across all five suppliers - no dock contention. Worker underutilisation (9.6%) is consistent with forklift-induced starvation, not a staffing problem. **Bottleneck chain.** Maintenance deferral reduced three forklifts to one-fifth speed, extending handling times by ∼5×, saturating the forklift tier and capping throughput at 1,352 packages instead of the expected 1,760.
Step 3 **Recommendation Agent** ***"What are your recommendations?"***	**Priority 1 - Restore forklift speed.** Forklifts 1, 2 and 3 are running at 1.0 kph - confirmed root cause. Restoring them to 5.5 kph is a single config update: `"FORKLIFTS.forklift1.speed_kph": 5.5, "FORKLIFTS.forklift2.speed_kph": 5.5, "FORKLIFTS.forklift3.speed_kph": 5.5` Expected improvement: handling times fall by ≈70%, relieving the storage-stage bottleneck and recovering the full inbound volume. No operational trade-off - this is a critical maintenance restoration. **Priority 2 - Balance supplier arrivals.** Staggering `supplier3` (t=80⟶90) and `supplier5` (t=140⟶180) pre-emptively reduces future dock pressure at minimal cost. **Priority 3 - Re-enable additional forklifts.** Even after speed restoration, forklift utilisation remains elevated. Re-enabling `forklift7` and `forklift8` provides capacity headroom for demand peaks.
Step 4 **Simulation Agent** ***"Apply Rec. #1 and re-run"***	The configuration was updated to restore `forklift1-forklift3` to 5.5 kph and the simulation was re-executed. The run completed with **1,760 packages** processed - recovering the full inbound volume throttled in the baseline. Average forklift handling time dropped to a normal 1.84-1.85 min across all active forklifts and total handling time was broadly balanced across the fleet. Average total package time was **206.36 min**, down from 242.75 min - a 14.9% improvement. *Simulation complete - ID:* `whsim.jsgza3`.
Step 5 **Comparison Agent** ***"Compare whsim.etlbs9 and whsim.jsgza3"***	Comparing baseline vs. optimised, forklift recovery is clear. The **average forklift wait time fell from 230.20 min to 193.43 min** (a 16% reduction), reflecting restored throughput capacity. Forklift utilisation held at 87.7%, indicating the speed restoration allowed the same tasks to complete faster rather than creating idle time. AGV utilisation rose from 20.4% to 23.8% and worker utilisation from 12.0% to 14.0%, both consistent with higher throughput unlocked downstream. Supplier discharge and dock metrics were unchanged - confirming the bottleneck was entirely within the forklift tier, with no inbound infrastructure involvement.

Table A.3. Representative agent trace - Scenario C: Dock Reduction + Arrival Clustering (Trial 1).

Scenario C - Dock Reduction + Arrival Clustering. One dock is taken offline for renovation, reducing capacity from 3 to **2 unloading docks**. A new supplier contract clusters three deliveries within a 20-minute window (t = 20, 30, 40 min), creating acute dock contention. Cascading worker idle windows misleadingly present as overstaffing in summary statistics. **Ground-truth fix:** Increase `NUM_UNLOADING_DOCKS` to 3 and/or stagger supplier arrival times.	
Step 1 **Simulation Agent**	

"Run the simulation"	The simulation completed with **1,710 packages** processed across 5 suppliers, 11 workers, 20 AGVs, 9 forklifts and 5 storage blocks. The average end-to-end package time was **159.24 min**. The clearest signal appeared in the supplier performance data: `supplier1` and `supplier2` experienced zero dock wait time and discharged in 75.55 and 60.55 min respectively, while `supplier3` waited **50.0 min** before a dock became available, `supplier4` waited 15.0 min and `supplier5` waited **42.5 min**. AGV and worker metrics appeared deceptively healthy - AGV wait time 0.0 min, worker utilisation ~12% - because both resource pools were sitting idle downstream of the dock bottleneck rather than being genuinely underutilised. *Simulation complete - ID:* `whsim.ydzep0`.
Step 2 **Auto-Analysis Agent** *"Run an automated analysis"*	**Executive summary.** A total of 1,710 packages were processed. Average end-to-end time: 159.24 min. Primary bottleneck: **dock contention** driven by clustered arrivals against a reduced two-dock infrastructure. Deceptively low worker (12%) and AGV (22%) utilisation are *symptoms* of the dock bottleneck starving downstream stages, not indicators of overstaffing. **Dock contention deep-dive.** Querying the knowledge graph for *"What is the dock wait time for each supplier?"* found `supplier3` waiting **50 min** and `supplier5` waiting 42.5 min. A follow-up query - *"What is the arrival time and discharge start time for each supplier?"* - confirmed three suppliers arriving within 20 minutes (t = 20, 30, 40), saturating both docks and forcing queuing. The knowledge graph flagged `dock_contention_signal = TRUE` and `clustering_flag = TRUE`; average supplier wait across all five arrivals was 21.5 min. **Other failure modes ruled out.** Forklift speed audit: all nine forklifts at 5.5 kph - no degradation. AGV capability: all 20 AGVs handle all three package types - no starvation. **Bottleneck chain.** Clustered arrivals against a two-dock facility create a supplier queue that forces workers and AGVs to sit idle downstream. Dock availability is the binding constraint for the entire pipeline.
Step 3 **Recommendation Agent** *"What are your recommendations?"*	**Priority 1 - Increase dock capacity.** Dock contention signal: `TRUE`. Clustering flag: `TRUE`. Average supplier wait: 21.5 min. Max wait: 50.0 min. These metrics collectively point to a structural capacity deficit. Config change: `"NUM_UNLOADING_DOCKS": 3` This eliminates queuing for `supplier3` and `supplier5`, reduces average dock wait to near-zero and frees the downstream worker and AGV resources currently idled by the bottleneck. **Priority 2 - Stagger supplier arrivals.** Even with a third dock, clustering at t = 20, 30, 40 min leaves little slack. Spreading arrivals to t = 10, 35, 55, 75, 105 min distributes dock load more evenly and provides resilience against volume spikes. **Priority 3 - Enable reserve forklifts.** Forklift utilisation reached ~89% at peak. Enabling `forklift10` and `forklift11` provides headroom once the dock bottleneck is resolved and full throughput is restored.
Step 4 **Simulation Agent** *"Apply Rec. #1 and re-run"*	The configuration was updated to `NUM_UNLOADING_DOCKS = 3` and the simulation was re-executed on the same clustered-arrival scenario. The run completed with **1,710 packages** processed. Supplier discharge dynamics improved measurably: the dock contention signal remained active (the clustered arrivals still stress three docks) but wait times were reduced - `supplier4`'s wait fell from 15.0 to 10.0 min. Average total package time was **158.64 min**, a marginal improvement consistent with applying only the top-priority fix while arrival clustering remained unchanged. AGV and worker utilisation ticked up slightly, reflecting the modest improvement in dock throughput. *Simulation complete - ID:* `whsim.nrqph0`.
Step 5 **Comparison Agent** *"Compare whsim.ydzep0 and whsim.nrqph0"*	Comparing baseline vs. optimised, improvement is measurable but partial - expected when only the dock count is changed and arrival clustering persists. Average supplier unload time improved marginally from 51.83 to 51.81 min. Worker utilisation rose from 19.4% to 20.0% and AGV utilisation from 32.8% to 34.0%, both consistent with fractionally higher downstream throughput. Forklift utilisation increased from 80.9% to 83.6%, indicating more packages are now reaching the storage stage. Notably, **AGV wait time increased from 10.23 to 24.45 min**, reflecting that the unblocked dock is now releasing packages faster than the rest of the pipeline can absorb - a downstream pressure resolved by also applying the stagger-arrivals recommendation. The comparison confirms dock capacity as the binding constraint and that combining Priorities 1 and 2 would yield a substantially larger joint improvement.