

Article

Exploring Decentralized Warehouse Management Using Large Language Models: A Proof of Concept

Tomaž Berlec , Marko Corn , Sergej Varljen and Primož Podržaj * 

Faculty of Mechanical Engineering, University of Ljubljana, 1000 Ljubljana, Slovenia;
tomaz.berlec@fs.uni-lj.si (T.B.); marko.corn@fs.uni-lj.si (M.C.); sergej.varljen@gmail.com (S.V.)

* Correspondence: primoz.podrzaj@fs.uni-lj.si; Tel.: +386-147-712-16

Abstract: The Fourth Industrial Revolution has introduced “shared manufacturing” as a key concept that leverages digitalization, IoT, blockchain, and robotics to redefine the production and delivery of manufacturing services. This paper presents a novel approach to decentralized warehouse management integrating Large Language Models (LLMs) into the decision-making processes of autonomous agents, which serves as a proof of concept for shared manufacturing. A multi-layered system architecture consisting of physical, digital shadow, organizational, and protocol layers was developed to enable seamless interactions between parcel and warehouse agents. Shared Warehouse game simulations were conducted to evaluate the performance of LLM-driven agents in managing warehouse services, including direct and pooled offers, in a competitive environment. The simulation results show that the LLM-controlled agent clearly outperformed traditional random strategies in decentralized warehouse management. In particular, it achieved higher warehouse utilization rates, more efficient resource allocation, and improved profitability in various competitive scenarios. The LLM agent consistently ensured optimal warehouse allocation and strategically selected offers, reducing empty capacity and maximizing revenue. In addition, the integration of LLMs improves the robustness of decision-making under uncertainty by mitigating the impact of randomness in the environment and ensuring consistent, contextualized responses. This work represents a significant advance in the application of AI to decentralized systems. It provides insights into the complexity of shared manufacturing networks and paves the way for future research in distributed production systems.



Academic Editors: Ugljesa Marjanovic, Slavko Rakic and Nenad Medic

Received: 24 February 2025

Revised: 13 May 2025

Accepted: 15 May 2025

Published: 20 May 2025

Citation: Berlec, T.; Corn, M.; Varljen, S.; Podržaj, P. Exploring Decentralized Warehouse Management Using Large Language Models: A Proof of Concept. *Appl. Sci.* **2025**, *15*, 5734. <https://doi.org/10.3390/app15105734>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: shared manufacturing; decentralized warehouse; large language models; multi-agent systems; resource optimization; capacity pooling; decision theory

1. Introduction

In the era of digital transformation, marked by the rise of servitization [1,2] and the evolution towards Industry 4.0 [3], modern global supply chains are increasingly characterized by decentralization, modularity, and dynamic interconnectivity [4,5]. Within this landscape, warehouse management emerges as a critical yet challenging component, often plagued by inefficiencies such as resource underutilization during off-peak periods, inventory imbalances during demand surges, and sluggish adaptation to real-time market fluctuations [6,7]. These issues, which align with the inefficiencies noted in traditional warehousing systems [8], highlight the pressing need for innovative, flexible, and cost-efficient warehouse management strategies.

Decentralized warehouse management presents a compelling solution, leveraging autonomous agents to optimize resource allocation and facilitate collaborative logistics.

This approach resonates with the production principles advocated by Schmidtke et al. [4], which emphasize the importance of decentralized, modular, and interconnected systems. A prime example of this paradigm is on-demand warehousing (ODW), where businesses can dynamically access third-party storage facilities to meet their fluctuating needs. Platforms such as Flexe and Ware2Go exemplify this model, enabling companies to convert fixed leasing costs into variable expenses and enhance their logistical agility [9]. Beyond benefiting small and medium-sized enterprises by reducing market entry barriers, ODW fosters collaborative networks that optimize resource utilization across the supply chain [10], supporting findings on the efficiency of collaborative warehousing [5].

Efficient resource allocation in ODW hinges on a classical optimization challenge akin to the secretary problem, where decisions must be made optimally under uncertainty as options arrive sequentially [11]. In this context, warehouse managers must decide whether to accept or reject storage requests without full knowledge of future opportunities, aiming to maximize profitability. While theoretically well explored, applying this concept in real-world settings is complicated by computational demands and unpredictable variables [12]. Recent studies further illuminate these challenges, offering advanced frameworks for online decision-making [12], analyzing platform-based resource matching [9], optimizing storage allocation with sequential approaches [13], and leveraging AI-driven tools to approximate optimal policies [14].

Recent breakthroughs in artificial intelligence, particularly the advent of LLMs, offer a promising avenue for navigating these challenges. LLMs, with their extensive pre-training on diverse corpora, encode rich semantic representations and emergent reasoning abilities, enabling them to interpret both structured inputs (e.g., capacity figures, timestamps, price quotes) and unstructured negotiation language without bespoke feature engineering [15]. In the context of warehouse management, LLMs can ingest real-time data streams, encompassing current warehouse utilization, incoming demand patterns, and dynamic pricing structures, and generate optimized storage decisions (to accept, reject, or forward offers) on the fly, thus eliminating the need for complex middleware or custom optimization solvers [16].

Moreover, by leveraging a stateless prompting strategy, each decision is framed as a self-contained query that includes all relevant background, the current state, and minimal history. This prompt-only approach sidesteps the substantial data collection, computational expense, and maintenance overhead associated with fine-tuning or training domain-specific models from scratch [14]. In practice, such a deployment yields sub-second inference latencies and controlled per-transaction costs while still achieving a decision quality that adapts to uncertainty and volatility, addressing many of the data-processing and standardization hurdles previously identified in AI adoption within supply chains [17].

Previous research on decentralized warehouse management has mainly focused on the use of multi-agent systems, game theory, and decision-theoretic models to improve resource allocation and logistical efficiency [4,5,10,18–21]. These studies demonstrate the potential of agent-based systems to simulate cooperation and competition between autonomous units while highlighting the challenges of optimal decision-making under uncertainty, such as the secretary problem [11]. On-demand warehousing platforms such as Flexe and Ware2Go have been investigated for their ability to convert fixed costs into variable costs and support SMEs in dynamic markets [9]. Recent advances in AI have introduced LLMs as potential decision engines in decentralized environments. Studies have shown that they are able to process structured and unstructured data and draw strategic conclusions without the need for extensive domain-specific training [14–16]. However, the previous literature has largely treated these elements—optimization, collaboration, and AI—separately. The present work bridges this gap by integrating LLMs into a decentralized game-theoretic warehouse sys-

tem, building on previous findings while demonstrating superior performance through simulation-based validation. Our contribution is thus located at the intersection of collaborative warehousing, AI-driven decision-making, and decentralized systems, providing a novel and practical architecture for collaborative production contexts.

1.1. Research Objectives

The aim of this study is to investigate the integration of LLMs in decentralized warehouse management systems and to evaluate their effectiveness in decision-making processes in competitive, resource-constrained environments. While decentralized logistics systems have been extensively studied in terms of agent-based modeling, the application of LLMs as autonomous decision-makers introduces a new dimension to the management of complexity, uncertainty, and resource optimization. This paper addresses the following research questions:

- RQ1: Can LLMs improve the effectiveness of decision-making in decentralized warehouse management systems compared to traditional random strategies?
- RQ2: How does the integration of a capacity pooling mechanism affect the performance of LLM-driven agents in resource allocation and warehouse utilization?
- RQ3: What is the impact of using LLMs on improving decision consistency and adaptability in environments characterized by uncertainty?

1.2. Research Contributions

This study makes the following important contributions:

Novel integration of LLMs in decentralized warehouse management: We propose a multi-agent system architecture that deploys LLMs as autonomous decision-makers capable of evaluating bids in real time without the need for fine-tuning or domain-specific solvers.

Simulation-based validation of LLM effectiveness: Using controlled experiments, we show that LLM-driven agents perform significantly better than random strategies in terms of warehouse utilization and profitability, especially in competitive, resource-constrained environments.

Evaluation of capacity pooling in decentralized logistics: We extend existing models by introducing and analyzing a capacity pooling mechanism and show how it improves decision quality and cooperation between agents.

Scalable architecture for shared production: The multi-layer system design (physical, digital shadow, organizational, and protocol layers) provides a flexible and scalable framework that can be adapted to real on-demand warehouse platforms.

Insights into AI decision-making under uncertainty: Our results show how stateless LLM input strategies can achieve consistent, context-aware decisions, contributing to a broader understanding of AI robustness in decentralized logistics systems.

1.3. Paper Structure Overview

The remainder of this paper is organized as follows: Section 2 introduces the theoretical background and related work, covering key concepts such as multi-agent systems, decision theory, game theory, and the role of LLMs in decentralized environments, as well as recent trends in shared manufacturing and collaborative warehousing. Section 3 outlines the proposed system architecture and describes its multi-layered structure—including the physical, digital shadow, organizational, and protocol layers—and details the design and goals of the autonomous parcel and warehouse agents. Section 4 explains the experimental setup, including the simulation environment, the implemented decision strategies, and the design of four different test scenarios. Section 5 explores the real-world applications of the proposed framework, focusing on the relevance to on-demand warehousing in e-commerce

and the potential for human–AI collaboration at the warehouse frontier. Section 6 reports the results of simulation experiments comparing the performance of LLM-driven agents with randomized baseline strategies in different configurations. Section 7 discusses the results in the context of the research objectives, analyzes the impact of capacity pooling and strategic decision-making, and compares the results with the current state of the art. Section 8 concludes with a summary of the main findings, a validation of the proof of concept, and an outlook on future research.

2. Theoretical Background

This section presents the theoretical background and relevant literature on decentralized warehouse management using artificial intelligence (AI). Key theories include multi-agent systems (MAS), game theory, decision theory, and artificial intelligence, with a focus on Large Language Models (LLMs). These concepts provide the framework for modeling autonomous agents, understanding their interactions, and improving decision-making in decentralized systems.

2.1. Multi-Agent Systems (MASs)

MASs involve multiple autonomous agents interacting within a shared environment to achieve individual or collective goals [18]. Agents in MASs are characterized by autonomy, social skills, reactivity, and proactivity [19]. The mathematical model of an MAS comprises equal agents (Equation (1)):

$$A = \{a_1, a_2, \dots, a_n\} \quad (1)$$

For each agent a_i , the state space S_i is defined, which is a set of possible states of the agent. The action space A_i is a set of actions available to the agent. The perception function $\text{Percept}_i : E \rightarrow O_i$ maps the environment E to the agent's observations O_i . The decision function (policy) $\pi_i : H_i \rightarrow A_i$, where H_i is the history or set of observations and states for the agent, determines the action selection. The state transition function $T_i : S_i \times A_i \rightarrow S_i$ defines how the agent's state updates based on its actions. Finally, the reward function $R_i : S_i \times A_i \rightarrow \mathbb{R}$ represents the agent's objective in terms of rewards.

The agents interact within a shared environment E characterized by its own state space S_E and state transition function T_E . The environment's state is updated based on the collective actions of all agents (Equation (2)):

$$T_E : S_E \times A_1 \times A_2 \times \dots \times A_n \rightarrow S_E. \quad (2)$$

After each discrete time step t , the next one occurs. Each agent a_i observes the environment using its perception function (Equation (3)):

$$o_i(t) = \text{Percept}_i(E(t)). \quad (3)$$

Based on its policy π_i and history $H_i(t)$, the agent selects an action (Equation (4)):

$$a_i(t) = \pi_i(H_i(t)). \quad (4)$$

The agents execute their actions, which affects the environment and possibly other agents. The agents then update their internal states according to their state transition functions (Equation (5)):

$$s_i(t+1) = T_i(s_i(t), a_i(t)). \quad (5)$$

Simultaneously, the state of the environment is updated based on the state transition function (Equation (6)):

$$E(t+1) = T_E(E(t), a_1(t), a_2(t), \dots, a_n(t)). \quad (6)$$

This mathematical framework is utilized in our system design to model the interactions and decision-making processes of agents within the decentralized warehouse management system.

2.2. Game Theory

A game theory models strategic interactions between rational agents [22]. It defines games with the help of players N , the strategy sets S_i for each player $i \in N$, and the payoff functions u_i .

A key concept is the Nash Equilibrium, a strategy profile $s^* \in S$ where no player can unilaterally improve their payoff (Equation (7)) by deviating from their strategy, formally expressed as

$$u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*), \quad \forall s_i \in S_i, \forall i \in N, \quad (7)$$

where s_{-i}^* denotes the strategies of all players except player i .

Game theory distinguishes between cooperative and non-cooperative games, providing insights into agent interactions in both competitive and cooperative scenarios, especially in decentralized systems.

2.3. Decision Theory

A decision theory models rational decisions under uncertainty and combines probability and utility theories [20]. It is a concept where an agent has a set of possible actions A , a set of possible states of the world Θ , and a utility function $U : A \times \Theta \rightarrow \mathbb{R}$ that assigns a real number to each possible outcome. The agent does not know which state $\theta \in \Theta$ will occur but has a probability distribution $P(\theta)$ representing their beliefs about the likelihood of each state. The agents choose actions to maximize the expected utility $EU(a)$ (Equation (8)):

$$EU(a) = \sum_{\theta \in \Theta} U(a, \theta) \cdot P(\theta). \quad (8)$$

where a rational agent chooses the action a^* (Equation (9)) that maximizes their expected utility:

$$a^* = \arg \max_{a \in A} EU(a). \quad (9)$$

The secretary problem illustrates the theory of optimal stopping for sequential decision-making, which can be applied in our system for agents deciding when to accept an offer and when to wait for better offers [21].

2.4. Artificial Intelligence in Decentralized Systems

Artificial intelligence (AI) encompasses a wide range of techniques and methodologies aimed at enabling machines to perform tasks that typically require human intelligence, such as perception, reasoning, learning, and decision-making [23]. In the context of decentralized warehouse management, AI plays a pivotal role by enabling autonomous agents to process complex, multidimensional data and make strategic decisions under uncertainty. AI-driven systems can interpret real-time data streams, including warehouse occupancy, demand patterns, and pricing structures, to optimize resource allocation and enhance logistical efficiency [16].

In multi-agent systems (MAS), as described in Section 2.1, AI facilitates the autonomy and social interactions of agents by equipping them with decision-making capabilities that adapt to dynamic environments. For example, AI can enable warehouse agents to evaluate incoming offers, negotiate terms, and coordinate services without centralized control, aligning with the principles of shared manufacturing [24]. The application of AI in such systems reduces the need for bespoke optimization solvers and supports scalable, flexible architectures suitable for on-demand warehousing platforms like Flexe and Ware2Go [9].

This study leverages AI to model intelligent agents capable of navigating competitive and collaborative scenarios, with a specific focus on advanced language processing techniques, as detailed in the following subsection.

2.5. Large Language Models

Large Language Models (LLMs), a specialized subset of AI technologies, have revolutionized Natural Language Processing (NLP) by enabling machines to understand and generate human-like language with unprecedented fluency and contextual awareness [25]. Built on the Transformer architecture, models like OpenAI's GPT-4 and Google's PaLM use a self-attention mechanism, defined in Equation (10), to process input sequences in parallel and capture long-range dependencies [15].

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \quad (10)$$

where Q , K , and V represent the query, key, and value matrices derived from the input embeddings, and d_k is the dimensionality of the key vectors.

Pre-trained on diverse corpora, including web texts and academic papers, LLMs encode rich semantic representations and emergent reasoning abilities through self-supervised learning [15].

In decentralized systems such as warehouse management, decision-making usually involves the evaluation of complex, multidimensional data under uncertainty. LLMs are particularly suitable for this area as they are capable of interpreting structured and unstructured inputs, including natural language prompts and symbolic data such as prices, capacities, and time constraints; generating contextual responses, making them effective agents for negotiation, scheduling, or exception handling; and learning and adapting to historical interaction data, enabling cumulative performance improvements without explicit reprogramming. For example, in our framework, LLMs are prompted with a structured format that includes background knowledge, the current system state, and previous decision outcomes. Based on these inputs, the LLM issues a decision (e.g., to accept, reject, or forward an offer) and thus acts as a strategic decision engine within a multi-agent system, as detailed in Section 3 (System Architecture).

Two primary prompting strategies are used in LLM applications: stateless prompting and stateful prompting. In stateless prompting, each interaction is independent, so the entire context (background (B), current situation (C), and historical data (S)) must be included in the prompt. This ensures reproducibility and is useful when system memory is not preserved between decisions. In contrast, stateful prompting allows the LLM to retain a kind of memory in which only incremental updates are provided. This is computationally more efficient and supports long-term strategic consistency, making it ideal for persistent agent roles in a distributed environment.

Mathematically, the prompt formulation can be expressed as shown in Equation (11):

$$\mathcal{P}_k = \begin{cases} \mathcal{P}(B_k \cup C_k \cup S_k) & \text{if Stateless Mode, for all } k, \\ \mathcal{P}(B_0 \cup C_0 \cup S_0) & \text{if Stateful Mode, at } k = 0, \\ \mathcal{P}(C_k) & \text{if Stateful Mode, for } k > 0. \end{cases} \quad (11)$$

In the stateless mode, the prompt explicitly includes the background B_k , current context C_k , and historical data S_k at each step, ensuring independence between iterations. In contrast, the stateful mode differentiates the initial prompt ($k = 0$) by explicitly including

B_0 , C_0 , and S_0 . For subsequent prompts ($k > 0$), B and S are implicitly inherited, requiring only the addition of C_k to maintain continuity across interactions.

Each method has its advantages and is suited to different scenarios. Stateless prompting is preferred in environments where reproducibility and independence between decisions are critical, while stateful prompting is useful for systems requiring long-term contextual awareness and cumulative decision-making. In this study, stateless prompting is employed to ensure reproducibility in the simulation environment (Section 4), achieving sub-second inference latencies suitable for real-time warehouse operations [14]. LLMs offer numerous advantages for AI-driven warehouse systems: natural language interaction, which enables collaboration between humans and agents; generalization across a wide range of tasks with a single model; and rapid deployment without extensive task-specific development. However, several challenges remain, including computational cost (LLMs are resource intensive to run at scale), lack of transparency (their decision-making process can be opaque), and sensitivity to prompt design (the quality of results is highly dependent on the quality of the prompt). LLMs improve decision-making by processing complex data and allowing agents to adapt their strategies in decentralized systems. However, challenges such as computational costs and model interpretability need to be addressed [26]. These limitations emphasize the need for careful system design and hybrid architectures where LLMs are paired with symbolic reasoning, structured protocols, or rule-based validation layers, as implemented in the proposed multi-layered architecture (Section 3).

2.6. Shared Manufacturing and Warehousing

Shared manufacturing extends the sharing economy to manufacturing and collaborative strategies in the product sharing market [27], where resources and capacity are pooled between participants to meet dynamic market demand [24]. This has resulted in an increased range of production platforms [28]. Blockchain facilitates the integration of services into a global market [29]. In warehousing, the concept of shared warehousing allows multiple companies to share storage space, optimizing logistics and reducing costs [10]. Inventory sharing pools resources to improve service levels and reduce costs [17]. Research has investigated the optimization of pooled inventories to mitigate fluctuations in demand [30]. However, collaboration in warehousing is still less studied compared to collaboration in transportation [5]. This theoretical background integrates MAS, game theory, decision theory, and AI to model and analyze decentralized warehouse management systems. MAS and game theory provide a framework for autonomous agent interactions, while decision theory guides rational decision-making under uncertainty. AI, especially LLMs, improves decision-making processes by enabling agents to adapt to the context. These theories, combined with the literature on collaborative manufacturing and inventory sharing, form the basis for the development of efficient decentralized warehouse systems.

3. System Architecture

The decentralized warehousing system is built upon a multi-layered architecture, with each of the following first four layers playing a specific role in system operation: the physical layer, the digital shadow layer, the organizational layer, and the protocol layer. In combination, these layers provide the infrastructure and data flows that enable the system to function seamlessly, as illustrated in Figure 1. In this layered structure, package agents and warehouse agents play the central role, autonomously managing storage services by interacting with and leveraging information from the four layers. The subsequent sections provide a detailed description of each layer's role and functionality, followed by an in-depth look at the agent design, which covers the states, decision-making processes, and interactions of the package and warehouse agents within this system.

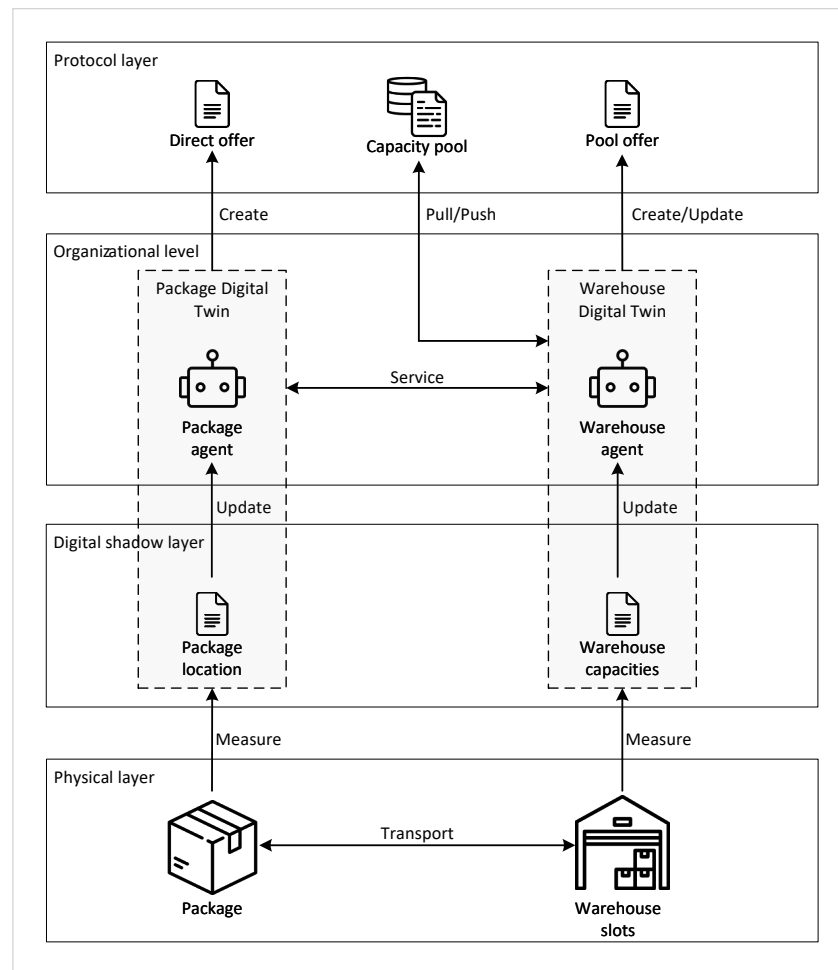


Figure 1. System overview of the decentralized warehousing network.

3.1. Physical Layer

The physical layer in the decentralized warehousing system represents the real-world, tangible components that form the foundation of the system's operations. This includes packages, warehouse slots, and transport processes. The physical layer manages the movement, storage, and spatial arrangement of goods, ensuring that every entity is properly accounted for in terms of its physical position and availability.

In this layer, each package is characterized by its location, as shown in Equation (12):

$$P_i = (x_i, y_i, z_i) \quad \forall i \in \{1, 2, \dots, N_P\} \quad (12)$$

where P_i represents the location of the i -th package, (x_i, y_i, z_i) denotes the 3D coordinates of the package within the system, and N_P is the total number of packages.

Similarly, each warehouse slot W_j is characterized by both its location and its occupancy status, as shown in Equation (13):

$$W_j = (x_j, y_j, z_j, S_j) \quad \forall j \in \{1, 2, \dots, N_S\}, \quad S_j = \{0, 1\} \quad (13)$$

where (x_j, y_j, z_j) are the 3D coordinates of the j -th warehouse slot, S_j is the occupancy status, where $S_j = 1$ if the slot is occupied and $S_j = 0$ if it is vacant, and N_S represents the total number of warehouse slots.

Transport processes are responsible for changing the states of packages and warehouse slots. They facilitate the movement of packages from one location to another, changing their location. The process operates between two points in time: the initial time t , when transport begins, and the final time $t + t_{\text{transport}}$, when transport is completed. Let $P_i(t)$

represent the location of package i at the starting time t , and $P_i(t + t_{\text{transport}})$ represent its location after the transport process is completed. Initially, the package's location is defined by Equation (14):

$$P_i(t) = (x_i(t), y_i(t), z_i(t)), \quad (14)$$

And, after the transport process (Equation (15)), the new location is

$$(P_i(t + t_{\text{transport}}) = (x_i(t + t_{\text{transport}}), y_i(t + t_{\text{transport}}), z_i(t + t_{\text{transport}})). \quad (15)$$

The transition can be represented as shown in Equation (16):

$$P_i(t + t_{\text{transport}}) = P_i(t) + (\Delta x_i, \Delta y_i, \Delta z_i) \quad (16)$$

where $(\Delta x_i, \Delta y_i, \Delta z_i)$ represents the change in the position of the coordinates of the package caused by the transport process. The changes in coordinates $(\Delta x_i, \Delta y_i, \Delta z_i)$ are defined as $(x_i(t + t_{\text{transport}}) - x_i(t), y_i(t + t_{\text{transport}}) - y_i(t), z_i(t + t_{\text{transport}}) - z_i(t))$.

The state of the warehouse slots also changes during the transport process, reflecting the movement of packages between locations. As a package is transported from one warehouse to another, the origin slot becomes vacant, and the destination slot becomes occupied. Let $W_j(t)$ (Equation (17)) represent the state of the origin warehouse slot at time t :

$$W_j(t) = (x_j(t), y_j(t), z_j(t), 1) \quad (17)$$

And let $W_k(t)$ represent the state of the destination warehouse slot at time t (Equation (18)):

$$W_k(t) = (x_k(t), y_k(t), z_k(t), 0) \quad (18)$$

$S_j(t) = 1$ indicates that the origin slot is occupied, and $S_k(t) = 0$ indicates that the destination slot is vacant. After the transport process, the occupancy of the slots changes as shown in Equation (19):

$$\begin{aligned} W_j(t + t_{\text{transport}}) &= (x_j(t + t_{\text{transport}}), y_j(t + t_{\text{transport}}), z_j(t + t_{\text{transport}}), 0) \\ W_k(t + t_{\text{transport}}) &= (x_k(t + t_{\text{transport}}), y_k(t + t_{\text{transport}}), z_k(t + t_{\text{transport}}), 1) \end{aligned} \quad (19)$$

This means that, after the transport process, the origin slot becomes vacant ($S_j(t + t_{\text{transport}}) = 0$), and the destination slot becomes occupied ($S_k(t + t_{\text{transport}}) = 1$).

The transport time $t_{\text{transport}}$ consists of three phases, as shown in Equation (20):

$$t_{\text{transport}} = t_{\text{load}} + t_{\text{transit}} + t_{\text{unload}} \quad (20)$$

The loading time (t_{load}) represents the period during which the package is prepared and placed onto the transport system. During the transit time (t_{transit}), the package physically moves between the origin and destination, where its coordinates change as a function of time. Finally, the unloading time (t_{unload}) accounts for the removal of the package from the transport system and its placement into the destination storage.

3.2. Digital Shadow Layer

The digital shadow layer is a part of the system's digital twin, providing a real-time, synchronized reflection of the physical layer. This layer mirrors the current states of packages and warehouse slots, allowing other system layers to access up-to-date information without directly interacting with the physical components. The state of the system in the digital shadow is updated continuously by measuring the real-world physical processes, ensuring accurate and timely representation of the system.

The package state in the digital shadow mirrors the physical layer. The location of each package at time t is reflected in the digital shadow, as shown in Equation (21):

$$\hat{P}_i(t) = (x_i(t), y_i(t), z_i(t)) \quad \forall i \in \{1, 2, \dots, N_P\} \quad (21)$$

where $\hat{P}_i(t)$ represents the digital shadow state of the i -th package and $(x_i(t), y_i(t), z_i(t))$ are the 3D coordinates of the package at time t , mirroring the real-world location. N_P represents the total number of packages.

Similarly, the warehouse slot state in the digital shadow reflects the physical state of the slot, including both its location and occupancy status, as shown in Equation (22):

$$\hat{W}_j(t) = (x_j(t), y_j(t), z_j(t), S_j(t)) \quad \forall j \in \{1, 2, \dots, N_S\} \quad (22)$$

where $\hat{W}_j(t)$ represents the digital shadow state of the j -th warehouse slot, $S_j(t)$ is the occupancy status of the warehouse slot at time t , with $S_j(t) = 1$ if it is occupied and $S_j(t) = 0$ if it is vacant, and N_S represents the total number of warehouse slots.

The digital shadow layer can also perform a higher level of data abstraction. The spatial part of each slot's location can be simplified by using slot IDs instead of precise 3D coordinates, as shown in Equation (23):

$$\hat{W}_j(t) = (\text{Slot}_j(t), S_j(t)) \quad \forall j \in \{1, 2, \dots, N_S\} \quad (23)$$

where $\text{Slot}_j(t)$ is the ID that abstracts the warehouse slot's spatial position and $S_j(t)$ represents the occupancy status, indicating whether the slot is occupied (1) or vacant (0).

Similarly, an abstraction of the spatial location of packages to the warehouse slot IDs can be performed. This means that, instead of tracking the exact 3D coordinates of packages throughout their journey, the system simplifies the process by associating each package with the ID of the warehouse slot where it is stored. This abstraction helps streamline the management of packages, focusing on the start and end points of the package's journey rather than continuously monitoring every movement.

When a package arrives at a specific warehouse slot, it adopts the ID of that slot as its location. Thus, the final state of the package is represented by the slot ID, as shown in Equation (24):

$$\hat{P}_i(t) = \text{Slot}_j(t) \quad \forall i \in \{1, 2, \dots, N_P\} \quad (24)$$

where $\hat{P}_i(t)$ represents the digital shadow state of the i -th package and $\text{Slot}_j(t)$ is the ID of the warehouse slot where the package is stored.

The digital shadow layer includes both detailed physical data and abstracted information. While the exact 3D coordinates of packages and warehouse slots are continuously tracked for precise monitoring, spatial abstraction simplifies management by mapping package locations to slot IDs. This duality ensures both real-time accuracy and simplified data which can be utilized by other layers.

3.3. Organizational Layer

The organizational layer is where the decision-making processes and interactions between agents take place. This layer hosts the package and warehouse agents, autonomously negotiating and executing services. A service represents the actions or resources requested by a package agent and provided by a warehouse agent. Two types of services are defined within this system: a storage service s_{storage} , which is responsible for storing a package in a warehouse slot, and a transport service $s_{\text{transport}}$, which involves transporting a package between warehouse slots, as shown in Equation (25):

$$\begin{aligned} s_{\text{storage}} &= (t_{\text{start}}, t_{\text{end}}, \text{Slot}_j) \\ s_{\text{transport}} &= (t_{\text{start}}, t_{\text{end}}, \text{Slot}_{\text{source}}, \text{Slot}_{\text{target}}, P_i) \end{aligned} \quad (25)$$

where t_{start} and t_{end} are the times that define the start and the end of the service execution, $Slot_j$ identifies the specific warehouse slot, $Slot_{source}$ and $Slot_{target}$ represent the source and target warehouse slots, and P_i is the ID of the package being transported.

In our current system, transport services are ignored under the assumption that the time required to execute storage services is significantly greater than the time required for the execution of transport services. As a result, transport times are considered negligible in comparison to storage times. Therefore, the transport service component does not have a major impact on the system's overall timing and execution of processes.

A storage service transitions through multiple states during its lifecycle, as shown in Equation (26):

$$s_{storage\ states}(t) = \begin{cases} \text{Negotiation,} & \text{if } t < t_{created} \\ \text{Scheduled,} & \text{if } t_{created} \leq t < t_{start} \\ \text{Active,} & \text{if } t_{start} \leq t < t_{end} \\ \text{Completed,} & \text{if } t \geq t_{end} \end{cases} \quad (26)$$

Initially, the service enters the Negotiation state (before $t_{created}$), where the parameters, such as the start time t_{start} , end time t_{end} , and warehouse slot, are agreed upon by the agents. After the negotiation is completed, the service transitions to the Scheduled state, which occurs before the start time, indicating that the service is planned but not yet active. Once the start time is reached, the service moves into the Active state, meaning the package is being stored in the designated warehouse slot between t_{start} and t_{end} . During this time, the warehouse slot is fully occupied by the package. Finally, when the end time is reached, the service transitions to the Completed state, at which point the package is removed from the slot, and the slot becomes available for future use.

3.4. Protocol Layer

The protocol layer plays a crucial role in facilitating the negotiation of services between agents in the organizational layer. While the agents are responsible for making autonomous decisions, the protocol layer provides the standardized frameworks and data structures that allow these agents to effectively communicate and negotiate service agreements, ensuring consistency across the system.

A direct offer is a data structure which is initiated by the package agent and sent to the warehouse agent as a formal proposal for service. The direct offer o_{direct} defines the terms under which the package agent requests a service and includes several essential parameters, as shown in Equation (27):

$$o_{direct} = (t_{start}, t_{end}, price, t_{expiry}) \quad (27)$$

where t_{start} is the proposed start time for the service, t_{end} is the proposed end time of the service, $price$ is the price proposed by the package agent for the service, and t_{expiry} represents the expiry time or deadline until which the offer is valid. Direct offers ensure that both the package and warehouse agents have a clear, standardized structure for negotiating the service terms.

The warehouse agent can respond to a direct offer in three distinct ways: by accepting, rejecting, or forwarding the offer. If the warehouse agent agrees to the terms set forth in the offer, the service is scheduled, and the package agent is informed that the service will proceed as requested. No formal contract is signed, but the scheduling of the service solidifies the agreement between the two agents. If the warehouse agent cannot fulfill the offer or disagrees with the proposed terms, the offer is rejected, and no further action is

taken. Alternatively, the warehouse agent may choose to forward the offer to the capacity pool, where other warehouses can review and potentially fulfill the request.

The capacity pool is a shared resource where warehouse agents can place unfulfilled direct offers for other warehouses to pick up. This system enables capacity pooling concepts, enabling warehouses to balance resources across the network. When a direct offer is forwarded to the pool, it becomes a pool offer o_{pool} , which retains the original offer's start time and end time but allows the warehouse agent to modify the price and expiry time. The price in the pool offer can be expressed as the original price from the direct offer plus a fee charged by the forwarding warehouse agent, as shown in Equation (28):

$$o_{pool} = (t_{start}, t_{end}, price + fee, t_{expiry_{new}}) \quad (28)$$

where t_{start} and t_{end} are the start and end times inherited from the original direct offer, $price$ is the price proposed in the original direct offer, fee represents the fee added by the forwarding warehouse agent, which can reflect their desired gain or added cost, and $t_{expiry_{new}}$ is the new expiry time set by the forwarding warehouse agent. The capacity pool P_{pool} functions as an aggregator of pool offers, centralizing unfulfilled service requests from individual warehouses into a shared resource, as shown in Equation (29):

$$P_{pool} = \{o_{pool_1}, o_{pool_2}, \dots, o_{pool_n}\} \quad (29)$$

Figure 2 shows a sequence diagram of the service negotiation process.

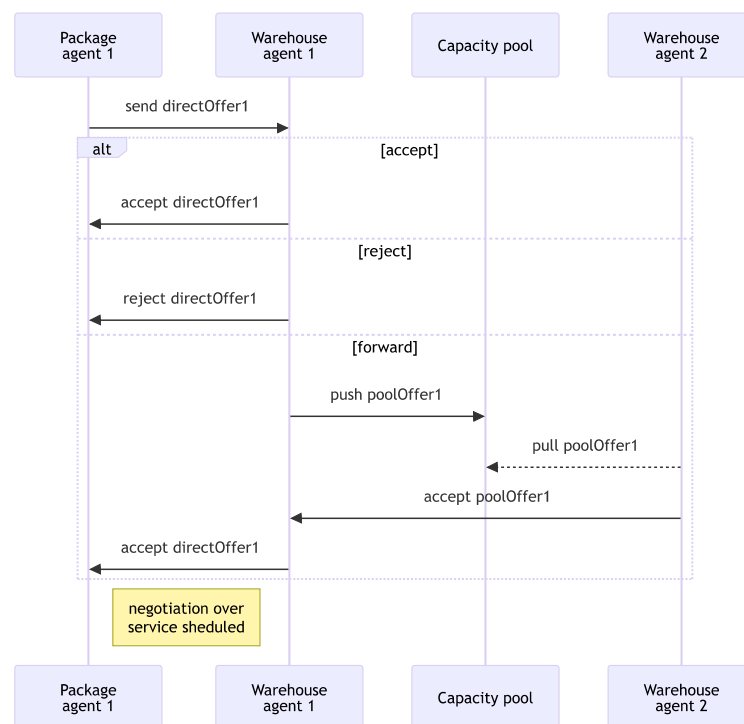


Figure 2. Negotiation process between package agent and warehouse agent.

PA_1 (Package Agent 1) initiates the process by sending $O_{direct\ 1}$ (directOffer1) to WA_1 (Warehouse Agent 1). The offer includes the proposed start and end times, price, and expiry time, as shown in Equation (30):

$$PA_1 \rightarrow WA_1 : o_{direct1} = (t_{start}, t_{end}, p, t_{expiry}) \quad (30)$$

Upon receiving the offer, WA_1 has three possible responses: to accept, reject, or forward the offer. If WA_1 accepts, it sends an acceptance response back to PA_1 , finalizing

the scheduling of the service $WA_1 \rightarrow PA_1 : \text{Accept}(O_{\text{direct1}})$. If the offer is rejected, WA_1 informs PA_1 of the rejection $WA_1 \rightarrow PA_1 : \text{Reject}(O_{\text{direct1}})$. Alternatively, WA_1 can forward the offer to the capacity pool C_1 as O_{pool1} (*poolOffer1*), modifying the price and expiry time while retaining the original start and end times, as shown in Equation (31):

$$WA_1 \rightarrow C_1 : o_{\text{pool1}} = (t_{\text{start}}, t_{\text{end}}, p + f, t_{\text{expiry_new}}) \quad (31)$$

This allows other warehouse agents, such as WA_2 (Warehouse Agent 2), to review and accept the offer from the pool. If WA_2 accepts the offer $WA_2 \rightarrow WA_1 : \text{Accept}(O_{\text{pool1}})$, WA_1 informs PA_1 that the service has been scheduled: $WA_1 \rightarrow PA_1 : \text{Service Scheduled}(O_{\text{direct1}})$. This formalizes the negotiation process, showing how agents exchange and process offers, with the capacity pool acting as a passive repository for unfulfilled offers.

In the negotiation protocol, the following inputs are gathered: the direct offer from the package agent (P1), which includes the start time, end time, price, and expiry time; and the internal state of the warehouse agent (W1), which comprises its current capacity and scheduled services. If the offer is forwarded, additional inputs emerge: the pool offer in the capacity pool (C1), which includes modified terms like the adjusted price and expiry time; and the internal state of another warehouse agent (W2), also consisting of its current capacity and scheduled services. Following these inputs, the outputs are produced, including W1's initial decision on the direct offer, which can be to accept, reject, or forward it (if forwarded, the pool offer is sent to C1), W2's decision on the pool offer, which can be to accept or ignore it (if W2 accepts, an acceptance message is sent from W2 to W1), and, finally, W1's ultimate response to P1, indicating whether the service is accepted or rejected.

3.4.1. Agent Design

In the decentralized warehousing system, package and warehouse agents make autonomous decisions based on a combination of perceptual data, derived insights, a decision process, and a defined action space. Perceptual data include real-time information such as storage availability and pricing, while derived insights are based on historical trends and performance metrics, offering a broader context for decision-making. Each agent uses a decision process to evaluate these data types and optimize its objective function. The action space outlines the available actions—such as sending, accepting, or forwarding offers—that agents execute to achieve their goals within the system. The agents balance real-time operations with strategic objectives through these components, ensuring efficient and optimized performance.

Package Agent Design

A package agent operates autonomously to ensure continuous storage for its package while minimizing the total cost associated with storage services. The agent relies on perceptual data (inputs) to make informed decisions and has a set of actions (outputs) available to achieve its objectives.

Perceptual data

The perceptual data $D_{\text{perceptual}_i}$ for package agent i include both the current occupied slot ID and the list of scheduled services, as shown in Equation (32):

$$D_{\text{perceptual}_i} = (\text{Slot}_i, S_i) \quad (32)$$

where Slot_i represents the abstracted identifier of the warehouse slot occupied by package agent i (see Equation (23)), and S_i is the list of scheduled services $S_i = [s_{i1}, s_{i2}, \dots, s_{in}]$. Each service s_{ij} in S_i is defined as per Equation (25) and includes details such as the start time $t_{\text{start}_{ij}}$, end time $t_{\text{end}_{ij}}$, and price p_{ij} .

Action space

The package agent's action space A_i encompasses all possible actions that the agent can perform to achieve its objectives within the decentralized warehousing system. In our system, the action space is defined by a single action, as shown in Equation (33):

$$A_i = \{\text{SendDirectOffer}\} \quad (33)$$

where SendDirectOffer represents the action of formulating and sending a direct offer O_{direct} to warehouse agents.

Decision Process

The primary objective of the package agent is to ensure continuous storage for its package while minimizing the total cost of storage. A penalty is applied for any gaps in storage, which can disrupt service continuity. The objective function for the package agent $\text{Objective}_{\text{package}}$ is defined as follows:

$$\text{Objective}_{\text{package}} = \sum_{k=1}^n p_k \cdot d_k + w \cdot G \quad (34)$$

where n is the total number of scheduled services, p_k is the price per unit time for the k -th storage service, $d_k = t_{\text{end}_k} - t_{\text{start}_k}$ is the duration of the k -th service, G represents the total duration of gaps in storage, penalizing any breaks in service continuity, and w is the penalty weight for gaps, reflecting their impact on the package agent's operations.

The package agent aims to minimize $\text{Objective}_{\text{package}}$ by negotiating for the lowest possible storage prices p_k and by ensuring there are no gaps ($G = 0$) between consecutive storage services in S_i . This objective function balances cost and service continuity, guiding the package agent's decision-making process.

Each package agent follows a specific decision-making process, leveraging the actions available in its action space to achieve its objectives. By continuously monitoring its perceptual data and scheduled services, the package agent proactively identifies when additional storage services are needed and initiates negotiations with warehouse agents to secure favorable terms.

The package agent's decision process is illustrated in Figure 3.

The process begins with the agent observing its storage needs. If no additional storage is required, the process terminates. If storage is required, the agent first checks if a direct offer has already been accepted. If the offer has been accepted, the agent adds a new storage service to its schedule and completes the process.

If the offer has not been accepted, the agent checks whether the offer has been rejected. If the offer has been rejected, the agent prepares a new direct offer and sends it to the warehouse agents. If the offer has not been rejected, the agent checks whether the direct offer has expired. If the offer has expired without being accepted or rejected, the agent prepares and sends a new direct offer.

Once a direct offer is accepted and the storage service is added to the schedule or a new offer is sent, the process is completed. This decision-making process allows the package agent to handle its storage requirements proactively, securing storage services as needed.

The package agent's decision-making process, shown in Figure 3, uses inputs including perceptual data on storage needs, current scheduled services, and the status of direct offers (accepted, rejected, or expired). The outputs include a decision to terminate if no action is needed, a new direct offer with parameters like start time, end time, price, and expiry time if the prior offer was rejected or expired, or a new storage service added to the schedule if an offer is accepted.

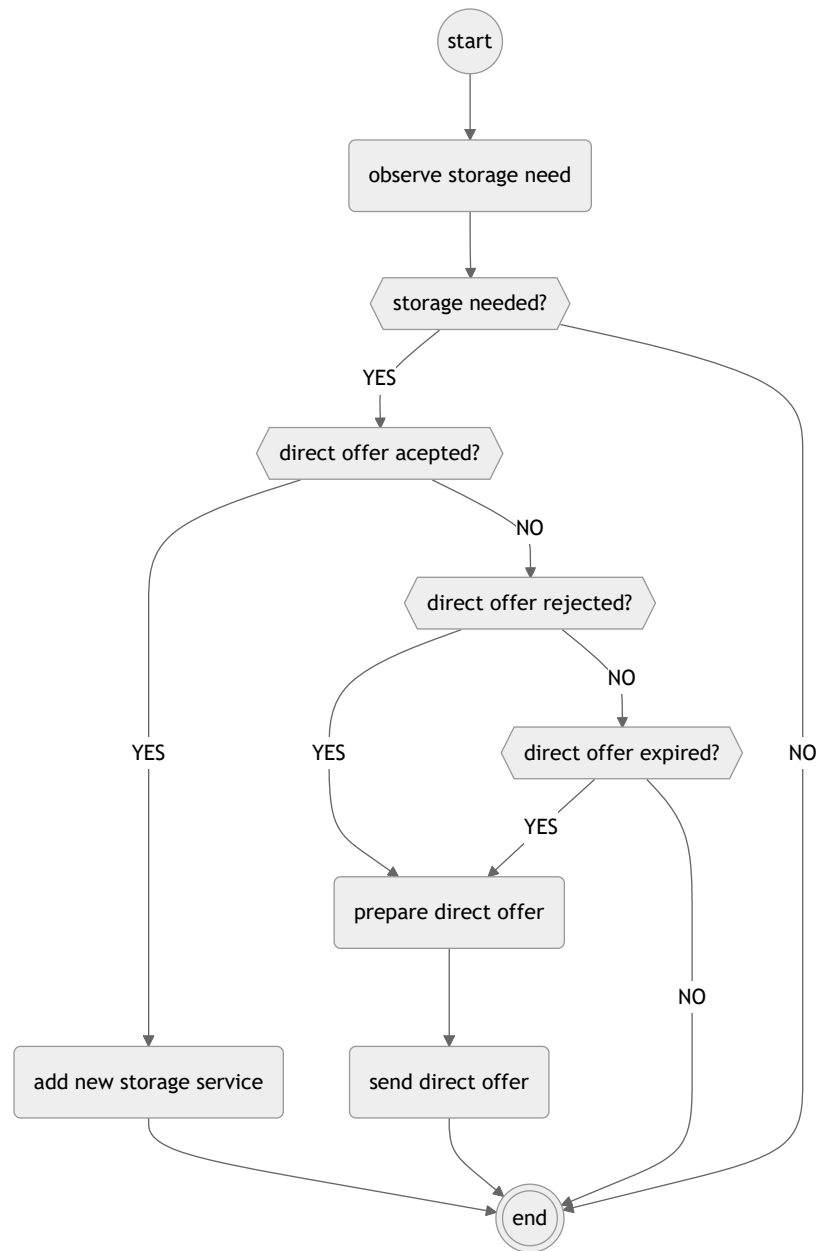


Figure 3. Package agent decision flow chart.

To delve deeper into the package agent's decision-making process, we focus on two main processes: observing storage needs and preparing direct offers. In the observing storage needs process, the package agent continuously monitors its perceptual data $D_{\text{perceptual}_i} = (\text{Slot}_i, S_i)$, particularly the list of scheduled services $S_i = [s_{i1}, s_{i2}, \dots, s_{in}]$. The agent checks for potential gaps in storage by examining the end times $t_{\text{end}_{ij}}$ and start times $t_{\text{start}_{i(j+1)}}$ of consecutive services. A gap exists if $t_{\text{end}_{ij}} < t_{\text{start}_{i(j+1)}}$, contributing to the total gap duration G . Additionally, the agent assesses if the time until the end of the last scheduled service $t_{\text{end}_{\text{last}}}$ is shorter than a predefined threshold Δt by checking if $T_{\text{current}} + \Delta t \geq t_{\text{end}_{\text{last}}}$. If either condition is met, the agent determines that additional storage is needed to prevent service disruptions.

Once the need for storage is identified, the agent proceeds to prepare a direct offer. The direct offer $O_{\text{direct}} = (t_{\text{start}}, t_{\text{end}}, p, t_{\text{expiry}})$ is formulated with the aim of minimizing costs and ensuring continuity. The start time t_{start} is set to the end time of the last scheduled service $t_{\text{end}_{\text{last}}}$ or the end time $t_{\text{end}_{ij}}$ of the service before a detected gap. The end time t_{end} is

determined based on the desired storage duration D_{desired} or set to the start time $t_{\text{start}_{i(j+1)}}$ of the next scheduled service to fill a gap. The proposed price p is set to minimize costs while being acceptable to warehouse agents, considering budget constraints and market rates. The offer expiry time t_{expiry} is set to encourage timely responses from warehouse agents.

By following these steps, the package agent proactively manages its storage needs, ensuring continuous storage and minimizing costs in alignment with its objective function.

Warehouse Agent Design

A warehouse agent operates autonomously to maximize revenue from storage services while efficiently managing its available capacity. The agent relies on perceptual data (inputs) to make informed decisions and has a set of actions (outputs) available to achieve its objectives.

Perceptual Data

The perceptual data $D_{\text{perceptual}_j}$ for warehouse agent j are defined as shown in Equation (35):

$$D_{\text{perceptual}_j} = (S_j, O_{\text{direct}_j}, O_{\text{pool}}) \quad (35)$$

where $(S_j = \{s_{j1}, s_{j2}, \dots, s_{jn}\})$ is a set of scheduled services that contains all the storage services already scheduled by the warehouse agent, $(O_{\text{direct}_j} = \{o_{\text{direct}_{1j}}, o_{\text{direct}_{2j}}, \dots, o_{\text{direct}_{mj}}\})$ is a set of direct offers that contains the offers received directly from package agents, and $(O_{\text{pool}_j} = \{o_{\text{pool}_{1j}}, o_{\text{pool}_{2j}}, \dots\})$ is a set of pool offers that contains the offers in the capacity pool, including those forwarded by other warehouse agents and by warehouse agent j itself.

These perceptual data are fundamental for the warehouse agent's decision-making, enabling it to evaluate incoming offers, manage its existing storage services, and potentially forward offers to other agents in the network.

Action space

The warehouse agent's action space A_j encompasses all possible actions that the agent can perform, as shown in Equation (36):

$$A_j = \{\text{AcceptDirectOffer}, \text{AcceptPoolOffer}, \text{RejectDirectOffer}, \text{ForwardDirectOffer}\} \quad (36)$$

The AcceptDirectOffer action is used when the agent agrees to provide the storage service as requested in the direct offer, signaling acceptance to the package or other warehouse agents. The AcceptPoolOffer signals the acceptance of a direct offer forwarded through the capacity pool. The RejectDirectOffer action communicates that the agent is not able or willing to accept the direct offer, without altering its current schedule or capacity. The ForwardDirectOffer action is used when the warehouse agent decides to forward the direct offer to the capacity pool, potentially with an added fee, allowing other warehouse agents to consider it. These actions serve as the outputs of the agent's decision process, facilitating interaction with the network while keeping the internal decision-making separate, which will be discussed in the next section.

Decision processes

The warehouse agent operates autonomously to maximize revenue from storage services and efficiently manage its available capacity. The agent fulfills two primary roles within the decentralized warehousing system: one as a provider of storage services and the other as a trader of offers in the capacity pool. In its provider role, the agent evaluates direct offers from package agents and decides whether to accept or reject them based on available capacity and potential revenue. In its trader role, the agent forwards unaccepted offers to the capacity pool, allowing other warehouse agents to accept the offers and earn a forwarding fee in return.

These two roles are reflected in the warehouse agent's objective function, which aims to maximize both the revenue from accepted storage services and the additional income

from trading offers in the capacity pool. The objective function is defined as shown in Equation (37):

$$\text{Objective}_{\text{warehouse}} = \sum_{i=1}^n \frac{p_i}{d_i} + \sum_{k=1}^m f_k \quad (37)$$

where p_i is the price accepted for the i -th service, d_i is the duration of the i -th service, $\frac{p_i}{d_i}$ represents the profit per unit time for each service, emphasizing high price and short duration, and f_k is the fee added to the k -th forwarded pool offer. The agent's objective function reflects its goal of maximizing profit by accepting a large number of profitable offers while also minimizing the impact on capacity. Therefore, the objective function prioritizes offers with high prices and short durations. Additionally, by offloading offers to other warehouses via the capacity pool, the warehouse can reduce its occupancy burden while generating additional income through fees.

Decision Process for Direct Offer

The warehouse agent's direct offer decision process is illustrated in Figure 4.

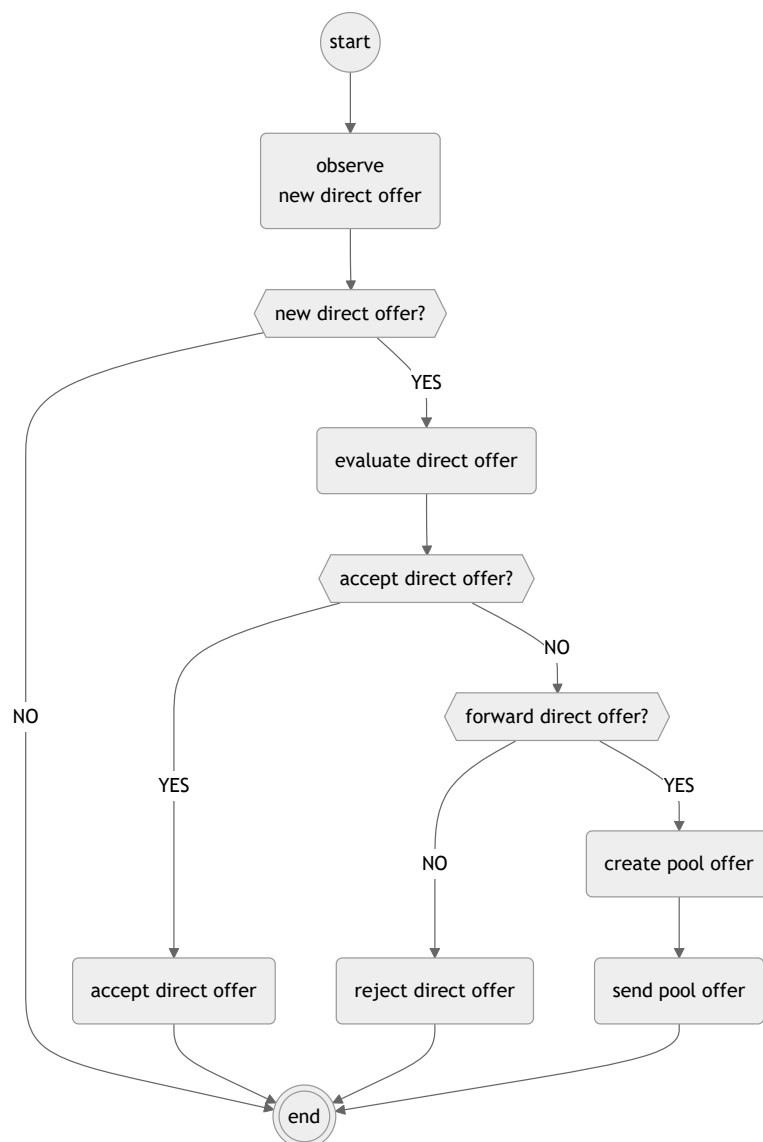


Figure 4. Warehouse agent direct offer decision process.

The process begins with the agent observing new direct offers from package agents. If no new direct offer is available, the process terminates. However, if a new direct offer is detected, the agent proceeds to evaluate the offer based on its internal criteria, such as available capacity and the revenue potential of the offer.

After evaluating the offer, the agent checks whether the direct offer is to be accepted or not. If the offer is acceptable, the agent accepts the offer, schedules the service, and the process completes. If the offer is not accepted, the agent checks whether the direct offer is to be forwarded to the capacity pool. If forwarding is chosen, the agent creates a pool offer by adding a fee and forwards it to the capacity pool, then completes the process. If the offer is neither accepted nor forwarded, the agent rejects the direct offer and the process terminates.

The warehouse agent's direct offer decision process, depicted in Figure 4, relies on inputs including new direct offers from package agents with parameters such as start time, end time, price, and expiry time, and the agent's internal state, encompassing available capacity and revenue goals. The outputs consist of a decision to terminate if no new offer exists, acceptance of the direct offer with a scheduled service if it is deemed acceptable, a pool offer with an added fee sent to the capacity pool if it is forwarded, or rejection of the direct offer if it is neither accepted nor forwarded.

This decision process allows the warehouse agent to make decisions regarding direct offers in a structured manner, balancing its storage capacity and revenue goals.

Decision Process for Pool Offer

The warehouse agent's pool offer decision process is illustrated in Figure 5.

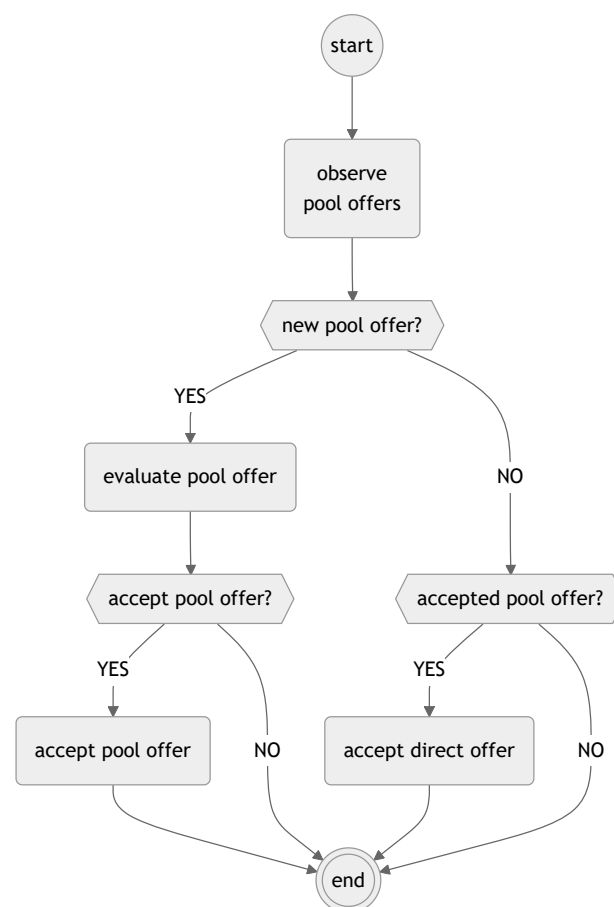


Figure 5. Warehouse agent pool offer decision process.

The process begins with the agent observing the pool of offers forwarded by other warehouse agents or their own previously forwarded offers. If no new pool offer is available, the agent checks if any of the previously forwarded direct offers have been accepted from the pool. If a pool offer has been accepted, the agent processes the acceptance and completes the process. If no new pool offer has been observed or accepted, the process terminates.

The warehouse agent's pool offer decision process, shown in Figure 5, uses inputs including new pool offers from the capacity pool with parameters such as start time, end time, modified price, and expiry time, the status of previously forwarded direct offers (accepted or pending), and the agent's internal state, encompassing current capacity and financial objectives. The outputs include a decision to terminate if no new or accepted pool offer exists, acceptance of a pool offer with a scheduled storage service if it is deemed profitable, or processing of an accepted direct offer from the pool, finalizing the service schedule.

If a new pool offer is detected, the agent evaluates the offer, considering whether it is in line with its current capacity and financial objectives. After evaluation, the agent checks whether to accept the pool offer. If the agent accepts the pool offer, the storage service is scheduled and the process is completed. If the pool offer is not accepted, the process terminates without making any changes to the agent's schedule.

This decision process allows the warehouse agent to handle offers in the capacity pool, enabling the agent to maximize its revenue by selectively accepting profitable offers or processing their own forwarded offers.

The warehouse agent's objective is to maximize revenue from storage services while managing available capacity. The warehouse agent can earn revenue through direct storage services and by forwarding offers with added fees to the capacity pool. The objective function for the warehouse agent is defined as shown in Equation (37).

Each agent follows a specific decision-making process, leveraging the actions available to them in their action space.

4. Experimental Setup

At the core of the experimental architecture, there is an application layer responsible for initializing the environment, creating agent entities, and managing the overall simulation flow. This logic is supported by a database layer, which ensures consistent storage, retrieval, and updating of all simulation data, including accounts, consumers, providers, and offers. These layers are complemented by a time management mechanism that advances the simulation's timeline in controlled increments, triggering event processing at precise intervals to enable deterministic and reproducible results.

4.1. Simulation Environment

The simulation environment was implemented using the following software stack:

- **Node.js:** v18.19.1
- **Operating System:** Ubuntu 22.04 LTS
- **Database:** MongoDB v6.0
- **Time Simulation Library:** @sinonjs/fake-timers v16.2.0

Simulations were executed on a local development machine equipped with an **Intel Core i7 processor, 16 GB RAM, and SSD storage**. The Node.js-based application layer controlled environment initialization, agent orchestration, and simulation flow. MongoDB persisted agent states, offers, and service data. Deterministic time progression was ensured using the @sinonjs/fake-timers library, allowing reproducibility of all simulation runs.

The complete source code, configuration files, and documentation are publicly available under a GNU General Public License v3.0 via the GitHub repository [31] under version 1.0.0.

4.2. Entities and Relations

Figure 6 shows a UML class diagram illustrating the primary entities used in the simulation. The diagram captures the key data structures—such as consumers, providers, and their associated services and offers—and the fundamental relationships between them. Following the diagram, each entity is described in more detail, outlining its general purpose and role within the system.

Each entity in the system defines a set of parameters that characterize its role and relationships. The Account entity holds a numeric balance, tracking available credit. The Consumer references one Account and maintains an array of services (identifiers to Service entities). Similarly, the Provider references an Account, stores an array of its services, and records a servicesLimit that constrains the maximum number of active services it can manage at once.

A Service specifies its state from a predefined set of lifecycle stages, stores arrays of offers (links to OfferDirect objects), and includes time-related parameters: startTimestamp, endTimestamp, and duration. It also holds references to the consumer and provider involved in that particular service and a count value for indexing or enumerating services.

OfferDirect defines commercial proposals by specifying seller and buyer accounts, a price, and an expiryTimestamp. Its state parameter indicates whether the offer is active, accepted, rejected, or expired, while the service link ties it to the corresponding Service.

If not accepted directly, an offer may be converted into an OfferCapacity, which introduces a fee parameter and can be placed in a capacity pool. OfferCapacity maintains a reference to the original offerDirect, along with its own seller, buyer, price, expiryTimestamp, and state parameters.

Lastly, PoolCapacity aggregates multiple offers (now OfferCapacity objects) and tracks which providers are associated with the pool. In this manner, the parameters of each entity define their basic structure, their links to other entities, and the essential timing, pricing, and state conditions that govern their participation in the system.

4.3. Behavioral Logic and Decision-Making Methods

The behavioral logic of providers and consumers is implemented in accordance with the agent design principles described in Section 3.4.1. In the experimental setup, two distinct approaches to decision-making were implemented: a random method, where actions are chosen probabilistically without deeper analysis, and an LLM-driven method, where an external Large Language Model provides context-sensitive guidance for more strategic decision-making.

4.3.1. Random Decision-Making Method

The random decision-making method enables agents to select actions based on predefined probabilities without strategic reasoning. Let p_{accept} , p_{reject} , and p_{postpone} denote the probabilities of accepting, rejecting, and postponing an offer, respectively. These probabilities satisfy the condition described in Equation (38):

$$p_{\text{accept}} + p_{\text{reject}} + p_{\text{postpone}} = 1 \quad (38)$$

The decision D is determined by a uniformly distributed random variable U over the interval $[0, 1]$ as shown in Equation (39):

$$D(U) = \begin{cases} \text{accept} & \text{if } U < p_{\text{accept}}, \\ \text{reject} & \text{if } p_{\text{accept}} \leq U < p_{\text{accept}} + p_{\text{reject}}, \\ \text{postpone} & \text{if } p_{\text{accept}} + p_{\text{reject}} \leq U \leq 1. \end{cases} \quad (39)$$

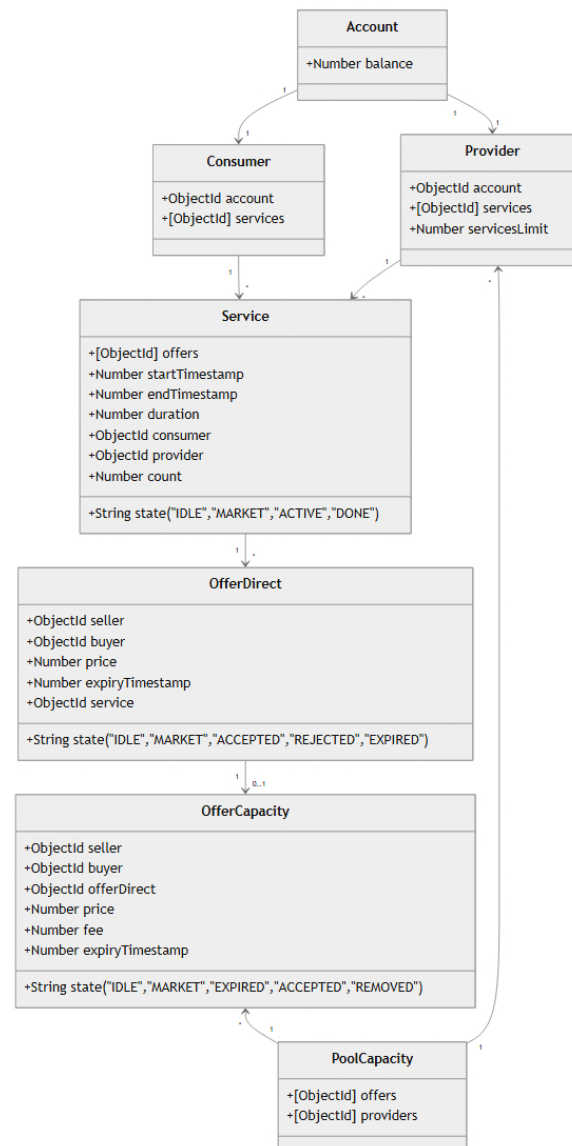


Figure 6. UML diagram of the entities of the simulation.

This mechanism ensures that each action is selected according to its associated probability, maintaining the stochastic nature of the agent's decision-making process without incorporating any strategic foresight or contextual analysis.

4.3.2. LLM-Driven Decision-Making Method

The LLM-driven decision-making method empowers agents to make informed and strategic decisions by leveraging the capabilities of a Large Language Model. Unlike the random method, which operates purely on predefined probabilities, the LLM-driven method integrates contextual understanding and historical data to optimize decision outcomes.

Let B denote the background context of the agent, which includes general system knowledge such as the operational principles of the on-demand warehousing system and predefined rules for the output generation. This background context provides a foundational understanding of the agent's decision-making process and, with the output definition, also a standardized way of communicating with the simulation environment. Let C denote the current context of the agent, encompassing real-time data such as current warehouse occupancy, details of incoming direct offers from consumer agents, and capacity offers posted in the capacity pool.

by other provider agents. Let S represent the historical data, including past direct offers, past capacity offers, previous decisions, and previous outcomes.

The agent formulates a prompt $\mathcal{P}(B, C, S)$ by combining background context B , the current context C , and the historical data S . This prompt is then sent to the LLM via an API call, resulting in a recommended decision D , as shown in Equation (40):

$$D = \text{LLM}(\mathcal{P}(B, C, S)) \quad (40)$$

where $D \in \{\text{accept, reject, forward}\}$.

This mechanism enables the utilization of the LLM as a decision optimization engine, allowing the agent to analyze and synthesize information from diverse data sources dynamically. Using background, contextual, and historical inputs, the LLM aims to generate adaptive and efficient responses to the complexities of the observed system, thereby improving operational outcomes through optimized strategies.

4.4. Experimental Data

The experimental setup was designed to evaluate the performance of warehouse agents within a decentralized warehousing system under competitive conditions. Four different simulation scenarios were implemented to evaluate the performance of the proposed shared storage system. The first two scenarios were without a capacity pool mechanism, while the second two were with a capacity pool mechanism. The simulation in this paper focuses more on a scenario where agents utilize a capacity pool mechanism to optimize resource allocation and profit generation.

The experimental environment consisted of five autonomous warehouse agents, each managing a warehouse with five slots. To evaluate the effectiveness of different decision-making strategies, four of the agents (Agents 1–4) were configured to use a randomized decision-making method that simulated non-strategic behavior. These agents responded to incoming storage offers by accepting, rejecting, or forwarding them based on predefined probabilities. In contrast, the fifth agent (Agent 5) used an AI-driven decision-making process based on an LLM, allowing it to make context-aware, strategic decisions based on historical and real-time data. This configuration allowed us to compare the performance of a single intelligent agent with that of multiple non-strategic agents in a competitive, decentralized environment. In Scenarios 3 and 4, a capacity pool mechanism was introduced that allowed agents to forward unaccepted offers to the shared pool where other agents could claim them for a fee—an additional layer of complexity and collaboration in the simulation. Table 1 summarizes the key parameters of the experiment.

Table 1. Scenario comparison.

Scenario	1	2	3	4
Capacity pool mechanism	Disabled	Disabled	Enabled, allowing offer forwarding	Enabled, allowing offer forwarding
Number of simulations per experiment	5	5	5	5
Warehouse capacity	1 storage slot per agent	5 storage slots per agent	1 storage slot per agent	5 storage slots per agent
Number of warehouse agents	5 agents (4 random, 1 AI-driven)	5 agents (4 random, 1 AI-driven)	5 agents (4 random, 1 AI-driven)	5 agents (4 random, 1 AI-driven)
Decision strategy	Random for 4 agents, AI-driven for 1 agent	Random for 4 agents, AI-driven for 1 agent	Random for 4 agents, AI-driven for 1 agent	Random for 4 agents, AI-driven for 1 agent
Number of consumers	5	25	5	25

The duration of each simulation was set to 100,000 time units, providing ample time for agents' strategies to evolve and their decisions to impact outcomes. Offers for storage services were generated with randomly assigned values between 1 and 10 tokens, and each storage service had a fixed duration of 1000 time units. These parameters created a dynamic and competitive environment for evaluating decision-making methods.

Agents with random decision strategy: Simulates non-strategic behavior using fixed probabilities to handle direct offers. Agents with a random decision strategy simulate non-strategic behavior by using fixed probabilities to consider direct offers. In simulations without a capacity pool mechanism, the probability to accept or to reject is 0.5. By using the capacity pool mechanism, in addition to acceptance or rejection, it is also possible to forward the offer. The outcomes and their probabilities are shown in Table 2.

Table 2. Random agent decision outcomes.

Outcome	Description	Probability (Pool)	Probability (No Pool)
Accept	Assigns a storage slot to the offer.	0.5	0.5
Reject	Declines the offer outright.	0.1	0.5
Forward	Sends the offer to the capacity pool.	0.4	0.0

LLM-driven warehouse agent: Implemented using OpenAI's ChatGPT (GPT-4 version). The experiments were conducted in March 2024. An agent was provided with a structured prompt containing contextual and historical data to make informed decisions. The testing was conducted exclusively in the stateless mode (Section 2.5), where the full context was included in every prompt without retaining information from previous interactions. Listing 1 is an example of the input prompt.

Listing 1. Example prompt for AI decision-making in stateless mode.

```

1 {
2   "context": {
3     "free_slots": 3,
4     "total_slots": 5,
5     "offer_history": [
6       {"status": "accepted", "price": 10},
7       {"status": "rejected", "price": 5}
8     ],
9     "current_offer": {
10      "price": 8,
11      "reward": 12
12    }
13  },
14  "instructions": "You are a warehouse agent operating in a shared
15    warehouse system. Your goal is to maximize profits while efficiently
16    utilizing storage slots. Based on the provided context, decide what
17    action to take. Options: ['accept', 'reject', 'forward']"

```

These prompts were constructed to maximize the agent's understanding of the current operational conditions (e.g., available storage slots and pricing history) and explicitly guide strategic decision-making. The structured design of these prompts ensured reproducibility and consistency across decisions. Moreover, by using stateless prompting, where each prompt includes all necessary context without reliance on previous interactions, the experimental setup avoided potential biases from sequence effects, thereby supporting robust comparative analyses between the randomized and LLM-based strategies.

4.5. LLM Prompting Approach

In this study, we utilize LLMs to enable autonomous decision-making in a decentralized warehouse management system. The LLMs are instructed via carefully designed prompts that provide the necessary context and information for the model to make informed decisions regarding warehouse operations.

4.5.1. Prompt Format

Each prompt is structured to include four key components: background information, historical data, the current situation, and the possible actions. The background information outlines the general context of the warehouse management task, including the goal of maximizing profit. The historical data provide a summary of past offers and decisions, such as the number of accepted, rejected, and forwarded offers, and the average price of accepted offers. The current situation details the warehouse's current state, including the number of occupied slots and the specifics of the current offer. Finally, the possible actions list the decisions the model can make: to accept, reject, or forward the offer.

The prompts are dynamically generated for each decision point, incorporating the latest state information from the simulation, as can be seen in Listing 1.

4.5.2. Parsing Logic

The LLM is instructed to respond with one of the following keywords: ACCEPT, REJECT, or FORWARD. The response text is parsed by searching for these keywords. If one of the keywords is found, the corresponding action is taken in the simulation. If none of the keywords is present, the system defaults to rejecting the offer to avoid unintended actions. This approach, combined with the explicit instructions in the prompt, minimizes the risk of misinterpretation by the model.

4.5.3. Settings

We used OpenAI's GPT-4 model for all experiments, specifically the version available as of March 2024. The temperature was set to 0.5 to balance creativity and consistency in the responses. The maximum number of tokens for each response was set to 50, which was sufficient for the model to provide a clear decision.

4.5.4. Handling Variability

Given the probabilistic nature of LLMs, responses can vary even for identical prompts. To mitigate the impact of this variability, we set the temperature to 0.5, which reduced extreme variations in the model's outputs while still allowing for some flexibility in decision-making. This approach ensured that the LLM-driven agents could make consistent and effective decisions across different simulation runs.

5. Real-World Application

The proposed framework for decentralized warehouse management has significant implications for modern logistics, especially in large-scale, dynamic, and demand-driven environments such as global e-commerce. The use of LLM-driven agents provides a practical and scalable approach to managing on-demand warehouses and improves decision-making in complex, distributed networks.

5.1. On-Demand Warehousing in E-Commerce

Retailers such as Amazon, Alibaba, and Zalando often face logistical bottlenecks during peak seasons such as Black Friday or the holidays. To alleviate these problems, on-demand warehousing enables dynamic scaling by external logistics providers. LLM-

controlled agents can autonomously handle contract negotiations, resource allocation, and service coordination in this network, significantly reducing the need for centralized orchestration. By using LLMs in this environment, agents can respond to real-time fluctuations in demand, availability, and pricing while making strategic considerations. Our simulation results support this capability and show how such agents improve utilization and profitability even in highly competitive environments.

5.2. Cost of Implementation

One concern with the adoption of LLM-based systems is the cost of implementation—especially when using commercial models such as GPT-4. Running Large Language Models (LLMs) can be resource intensive, particularly when they are deployed at scale. To address implementation costs, we are exploring strategies including smaller models (using distilled or lightweight versions of LLMs that retain performance while reducing computational overhead); prompt optimization (minimizing token usage in prompts to lower processing costs); and cloud solutions (leveraging dynamic cloud-based infrastructure that scales resources based on demand, ensuring cost efficiency). These approaches will help make the system more financially viable for real-world use. In practice, however, the use of LLMs in decentralized warehousing need not be prohibitively expensive:

- The cost of inference can be minimized by using smaller, open-source LLMs or domain-specific models tailored to logistics tasks.
- Stateless prompting, as used in our simulation, provides compatibility with serverless architectures and API-based deployment, ensuring low computational overhead per transaction.
- Edge deployment of lightweight models (e.g., LLaMA or Mistral variants) can localize decision-making and further reduce latency and cloud costs.

In addition, the return on investment can be justified by better warehouse utilization, less negotiation effort, and less dependency on central IT infrastructure.

5.3. Real-Time Performance Aspects

Real-time responsiveness is critical in warehouse environments. Although LLMs are not traditionally designed for low-latency applications, several factors make them suitable here:

- Prompt processing times for smaller models are typically less than 500 ms, which is acceptable for scheduling, contract evaluation, and strategic proposal processing.
- Batch processing of similar requests can improve throughput.
- Session caching in stateful prompting modes enables incremental decision updates, reducing the size of prompts and speeding up response times.

In high-decision-frequency environments, hybrid architectures can be used where LLMs are supplemented by faster rule-based systems for routine or low-risk tasks.

5.4. Scalability of Deployment

Scalability is a key advantage of our proposed architecture. Since agents operate autonomously and communicate via standardized offers and responses, new nodes (e.g., warehouse providers) can be integrated with minimal configuration. In addition,

- LLM agents do not require centralized control or a global system state.
- The protocol layer and capacity pooling mechanisms are inherently decentralized and support horizontal scalability across networks of varying size and complexity.
- Simulation results show that LLM agents maintain consistent decision quality even in resource-constrained constellations. For the system to succeed in a real-world setting, it must handle increased loads effectively. We propose enhancing scalability through

a distributed architecture (deploying multiple LLM instances in parallel to manage different segments of the warehouse network) and load balancing (implementing efficient resource allocation to maintain performance under heavy demand).

These strategies will enable the system to scale seamlessly as operational needs grow and make the architecture adaptable for both SMEs and large enterprises.

5.5. Explainability and Trust in LLM Decisions

One challenge with LLM-based systems is the interpretability of their decisions. Unlike rule-based algorithms, LLMs operate as black boxes, which raises concerns about trust and verifiability. To solve this problem, the prompt design in our system includes explicit instructions and structured input formats that improve the consistency and traceability of decisions. In addition to that, the output masks can be extended with justification tokens (e.g., “Accepted because...”), which increases transparency for operators. Logging of prompts and post-decision outputs can be used to review agent behavior and refine prompting strategies. In practice, confidence can be built by comparing LLM decisions to known heuristics, visualizing historical performance, and performing human validation in the loop when needed. The “black box” nature of LLMs poses challenges for trust and accountability. To improve explainability, we are considering audit trails (logging all prompts and responses to provide a transparent record of decision-making) and attention visualization (using techniques to highlight how the model focuses on specific inputs, offering insights into its reasoning). These steps will make the LLM’s decisions more interpretable, fostering trust among users and stakeholders.

5.6. Human–AI Collaboration at the Warehouse Edge

In addition to large companies, this model can also give small and medium-sized warehouse operators the opportunity to monetize unused capacity and participate in broader logistics networks. These operators, who often do not have sophisticated AI systems, can benefit from LLM-driven agents embedded in a common platform. These agents handle negotiation, planning, and contract execution in natural language or in simplified structured formats, lowering the barrier to entry and minimizing the need for advanced technical knowledge. In addition, the use of collaborative robots (cobots) and IoT edge devices in these warehouses can be improved through LLM integration. For example, an LLM-supported system can orchestrate the distribution of tasks between human employees and autonomous systems, ensuring efficient coordination while responding dynamically to changes in inventory or transportation availability.

5.7. Future Extensions and Integration

In the future, this system could be integrated with blockchain-based smart contracts to enable trustless transactions between agents, and with predictive analytics tools to forecast demand patterns and pre-position inventory. Furthermore, by training LLMs on domain-specific historical logistics data—such as data on delivery delays, return rates, or supplier reliability—agents can make increasingly sophisticated and informed decisions. Overall, the use of LLM-driven agents in decentralized warehousing represents a scalable and robust solution that is in line with the goals of Industry 5.0: improving the collaboration between humans and AI, sustainability, and adaptability of the system.

6. Results

As described in the experimental setup, four simulation scenarios were implemented to evaluate the performance of the proposed shared storage system. Scenarios 1 and 2 varied warehouse capacities (one vs. five units), while Scenarios 3 and 4 introduced a capacity

pooling mechanism, allowing agents to forward unaccepted offers. This structure enabled a systematic assessment of how warehouse size and pooling affect agent performance and resource utilization.

In Scenario 1, each agent operated a single storage slot without the pooling mechanism. The LLM-driven AI agent outperformed random counterparts, winning three out of five simulations and earning an average of 249 tokens, compared to 232 for the random agents. As shown in Figure 7, this success can be attributed to more consistent and complete warehouse utilization. This result indicates that the AI agent strategically manages limited resources to maximize efficiency.

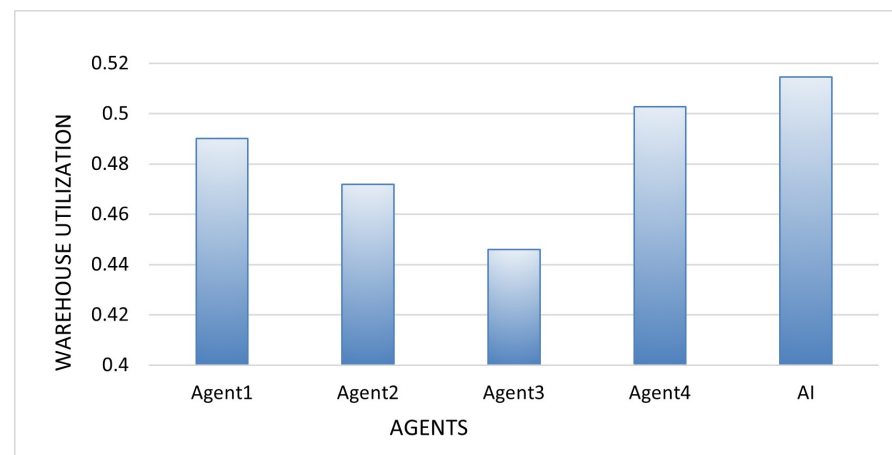


Figure 7. Average warehouse utilization simulation Scenario 1.

In Scenario 2, warehouse capacity was increased to five slots. Here, the AI agent won only two of five games—equal to Agent 4. The AI reserved later slots for potentially higher-value offers, but this strategy resulted in underutilizing slots 3–5, as shown in Figure 8. The mismatch between offer frequency and the AI’s expectations illustrates the limitations of the stateless prompt strategy in more complex configurations.

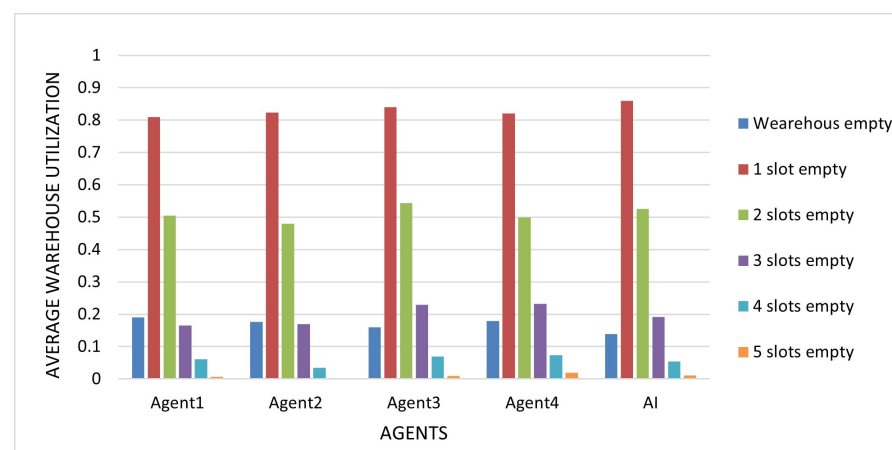


Figure 8. Average warehouse utilization simulation Scenario 2.

Scenario 3 introduced the capacity pool while retaining the single-slot configuration. The AI agent won four out of five games and led in token earnings. With the ability to forward less favorable offers, the AI reduced reliance on chance, achieving superior utilization (Figure 9). This performance reinforces and provides strong evidence that capacity pooling significantly enhances decision quality and consistency.

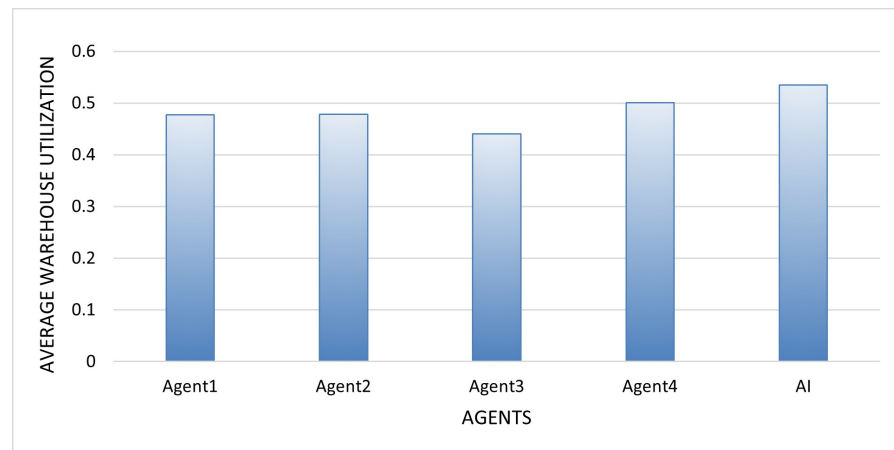


Figure 9. Average warehouse utilization simulation Scenario 3.

Scenario 4, the most complex and realistic configuration, combined five storage slots with the pooling mechanism. The AI again won four out of five games and earned the most tokens (Figure 10). Although storage utilization was similar across agents (Figure 11), the AI achieved greater token efficiency, selectively forwarding, accepting, and rejecting offers.

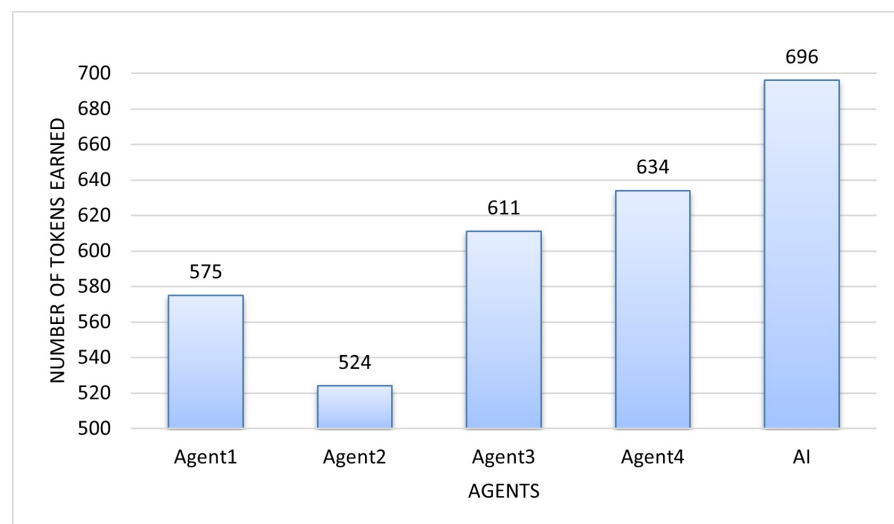


Figure 10. Average number of tokens earned.

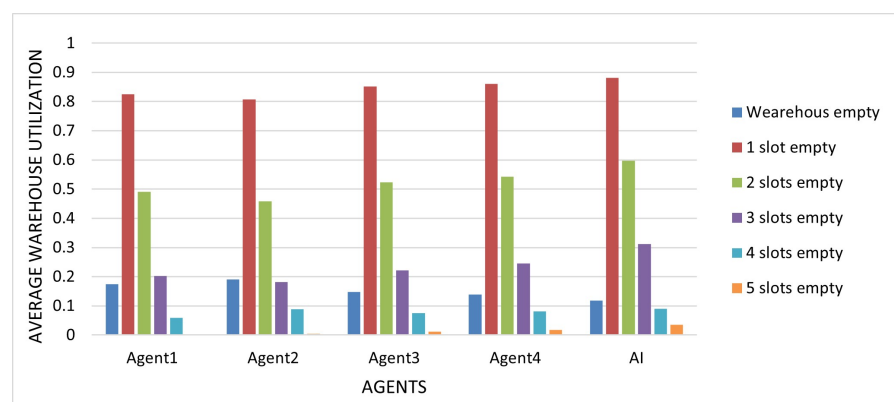


Figure 11. Average warehouse utilization.

As shown in Figure 12, the AI submitted the highest number of offers to the pool and achieved the most successful submissions. Although it accepted fewer offers from the pool than other agents (Figure 13), it optimized the results through a higher token-per-offer ratio.

These findings confirm that AI effectively adapts to complex environments by making context-sensitive, profit-optimized decisions.

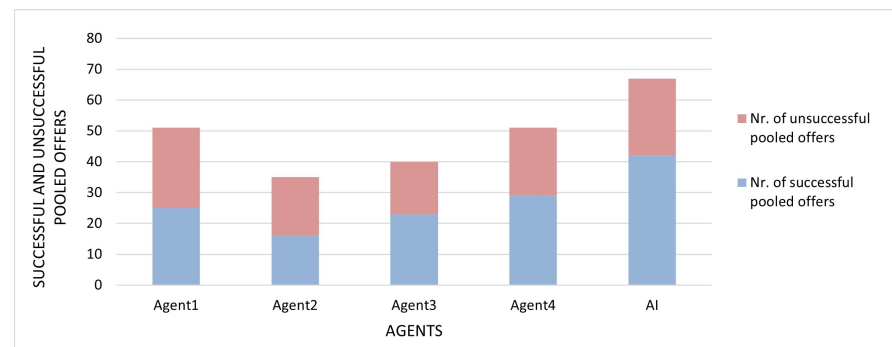


Figure 12. Successful vs. unsuccessful pooled offers.

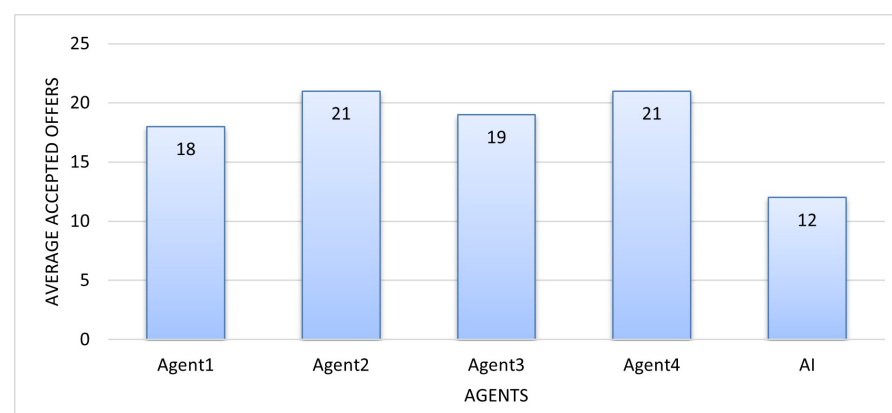


Figure 13. Average accepted offers from the pool.

The decision-making process is illustrated in Figure 14. The AI consistently preferred submitting to the pool over rejecting offers. Even in its worst round, it achieved the highest average tokens per accepted offer (3.9) compared to other agents (3.7 and 3.4), showcasing robustness under uncertain conditions.

Despite having a larger storage capacity, the AI opted to accept fewer offers from the pool, likely recognizing that keeping the storage less occupied would maximize its relative token yield. Therefore, AI managed to achieve an average of over four tokens per accepted offer, which is the highest score across all scenarios. This exceptional performance is likely due to the complexity of the scenario, which allowed for the largest number of possible decision combinations, potentially leading to optimal outcomes. The AI employed a combination of efficient decisions, including accepting a higher number of offers, which is reflected in its optimal storage utilization, where the AI fully utilized all five storage slots better than the other players. Figure 14 shows the players' decision-making process in relation to rejected offers.

Once again, the AI recognized that the expected profit from submitting offers to the pool was consistently higher than from rejecting them. By understanding this ratio and the constant balance between successful and unsuccessful submissions, the AI achieved greater earnings than the other players. Interestingly, even in its worst game, the AI still managed to achieve the highest number of tokens per accepted offer, which was 3.9, in comparison with the others, who achieved 3.7 and 3.4. The lower performance in this game was due to the limited number of accepted offers and consequent poorer storage utilization (Figure 15), as evidenced by the suboptimal utilization of the later storage slots. The AI had

almost half the capacity filled in its third storage slot compared to other players, which also held true for the fourth and fifth storage slots in this game.

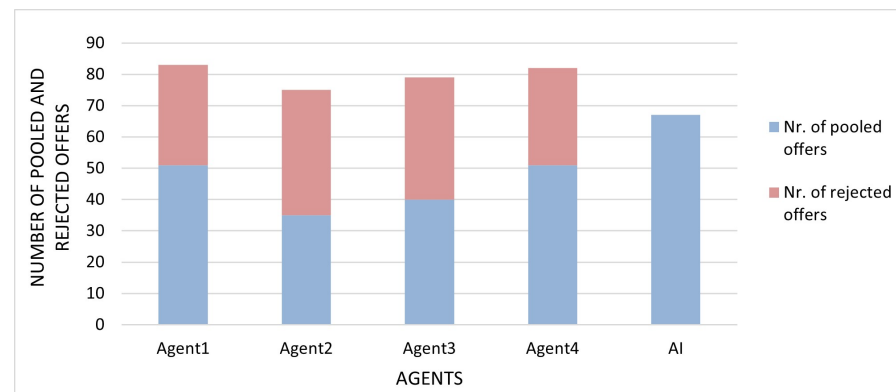


Figure 14. Number of pooled and rejected offers in the best AI game.

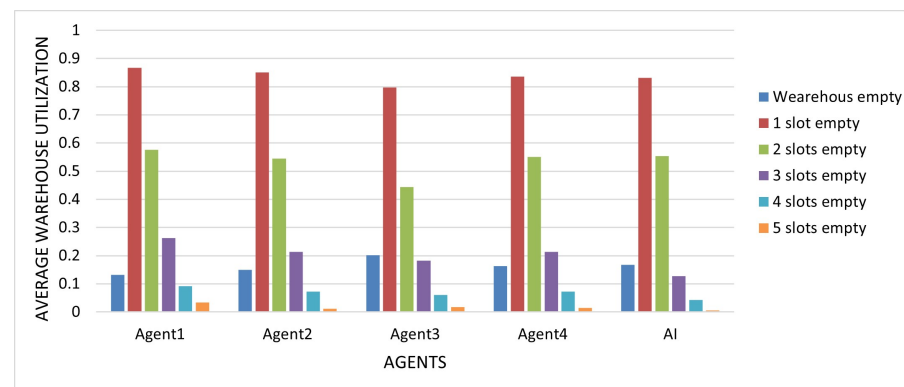


Figure 15. Storage utilization in the worst AI game.

7. Discussion

The discussion section consists of three main topics: analysis of the simulation results, revision of the research objectives, and comparison of the findings to the state of the art.

7.1. Analysis of Simulation Results

The simulation experiments across four distinct scenarios demonstrated the effectiveness of integrating LLM-driven agents into decentralized warehouse systems. In scenarios without capacity pooling, the LLM agent consistently outperformed its random counterparts by achieving higher warehouse utilization and greater overall profitability. Specifically, in Scenario 1 (single slot, no pooling), the LLM agent won the majority of simulation rounds and achieved superior warehouse utilization, demonstrating its ability to strategically allocate limited resources. However, in Scenario 2, where warehouse agents had five storage units and no access to pooling, the AI agent's performance was less dominant. Here, the AI employed a strategy of reserving capacity for higher-value offers in later slots, which ultimately led to underutilization. This reveals a limitation in the AI's predictive ability when operating in isolation from collaborative mechanisms. The introduction of the capacity pool in Scenarios 3 and 4 significantly improved AI performance. In Scenario 3 (single slot with pooling), the LLM agent exhibited enhanced decision-making by leveraging the pool to off-load or forward offers efficiently, outperforming all competitors. Scenario 4 (multi-slot with pooling) provided the richest environment for testing strategic complexity. The LLM agent won most simulations, optimized its token yield per offer, and strategically submitted the highest number of successful pool offers while still accepting fewer low-value offers than

its peers. These findings underscore the LLM agent's capacity to maximize profit not just through utilization, but also by optimizing offer evaluation and timing.

7.2. Relevance to Research Objectives

This study addressed three key research questions related to the use of LLMs in decentralized warehouse management. The results relating to each question are discussed in detail below:

- RQ1: Can LLMs improve the effectiveness of decision-making in decentralized warehouse management systems compared to traditional random strategies? The simulation results show that LLM-driven agents consistently perform better than agents using random strategies in terms of both warehouse utilization and total number of tokens earned. This was evident in all four simulation scenarios. The LLM agent showed a clear ability to reason about the quality of offers and make selective decisions that led to higher profitability. Even in situations with minimal storage capacity (e.g., Scenario 1), the LLM agent avoided suboptimal decisions and maintained better resource utilization over time. These results show that LLMs can act as effective autonomous agents capable of strategic thinking in a competitive multi-agent environment—something that random strategies cannot replicate.
- RQ2: How does the integration of a capacity pooling mechanism affect the performance of LLM-driven agents in resource allocation and warehouse utilization? The addition of a capacity pooling mechanism (Scenarios 3 and 4) significantly improved the effectiveness of the LLM agent. The agent used this mechanism to pass on offers that it could not fulfill, thereby adding value through pool fees or holding back capacity in anticipation of better opportunities. The pooling mechanism provided flexibility and reduced the risks associated with tying up resources prematurely. Compared to scenarios without pooling, the AI agent achieved higher average token revenue and had a higher success rate for forwarded offers. This suggests that pooling mechanisms are not only beneficial system wide, but are particularly advantageous when combined with intelligent agents that are able to draw conclusions about future states.
- RQ3: What is the impact of using LLMs on improving decision consistency and adaptability in environments characterized by uncertainty? The AI agent showed strong adaptability to fluctuating demand and unpredictable supply flows. Unlike the random agents, who reacted statically, the LLM was able to recognize favorable patterns and adapt its strategy over time, for example, by selectively rejecting or forwarding offers based on occupancy, price trends, and supply history. Even in the most complex scenario (Scenario 4), where resource constraints, competition, and dynamic pricing were present, the LLM agent made decisions that aligned immediate reward with long-term strategic positioning. Its decisions were also more consistent and explainable as the LLM relied on structured prompts to place each offer in the context of the overall system state. This behavior demonstrates that LLMs can maintain a high level of decision quality even when exposed to noisy, dynamic, or incomplete information—a key requirement in real-world decentralized logistics.

To summarize, the results not only confirm that LLMs improve decision performance, but also show that their effect is enhanced when supported by mechanisms such as capacity pooling. Moreover, LLMs are well suited to operate in uncertain and decentralized environments where conventional heuristics or rigid policies may fail.

7.3. Limitations of the Proof of Concept

As a proof of concept, our study has several important limitations that should be addressed in future work: Simplified simulation environment. We assume negligible transport times, uniform offer distributions, and a small fixed number of agents and slots. Real-world warehouses exhibit far more complex dynamics—variable transit delays, non-stationary demand, handling constraints, and stochastic failures—that are not captured here. Single LLM and prompting mode. We evaluated only one pre-trained model via stateless prompting. We did not explore alternative architectures, smaller on-premise models, stateful prompting, or systematic prompt tuning. Different choices could yield different trade-offs in accuracy, latency, and cost. Cost, latency, and scalability. While our API-based setup achieved sub-second decision times in simulation, larger-scale or higher-frequency deployments may incur prohibitive inference costs and delays. We did not benchmark throughput under concurrent load or consider hybrid architectures with rule-based fallbacks. Opaque decision logic. LLM outputs remain largely black box. We did not integrate explainability techniques, rule-based sanity checks, or human-in-the-loop validation to audit or justify agent decisions—features often required for operational use. Limited baseline comparison. We compared the LLM agent only against a randomized strategy. Without quantitative evaluation against rule-based heuristics, optimization solvers, or learned policies, it remains unclear how our LLM’s performance gains compare to existing methods. No real-world data validation. All results were derived from synthetic game-based simulations. We have not yet tested the setup on real warehouse telemetry, historical bids, or live market data, so robustness and generalizability remain to be demonstrated. Addressing these limitations—by enriching the simulation, evaluating multiple models and prompting strategies, measuring real-world performance, adding explainability layers, and expanding baseline comparisons—will be the focus of our ongoing and future research.

7.4. Comparison with State of the Art

Compared to traditional agent-based or rule-based decision strategies in collaborative logistics systems, this study introduces a significant innovation through the use of LLMs. While prior works such as those of Wang and Yue [12] and Jamili et al. [12] focus on the optimization of inventory pooling through mathematical models or simulations, our approach leverages pre-trained language models to navigate a more flexible and unpredictable environment. Unlike deterministic rule sets, LLMs operate using probabilistic reasoning based on historical and contextual prompts, enabling adaptability in real time. Moreover, many existing studies on warehouse collaboration assume cooperative agent behavior or centralized orchestration. In contrast, our work places autonomous agents in a decentralized and competitive setting, emphasizing individual profit maximization and market-driven negotiation. The use of LLMs enables these agents to simulate strategic reasoning, such as offer timing and value assessment—functions rarely addressed in the current warehouse management literature. Finally, in comparison to digital twin frameworks or decision support systems that rely on structured data and predefined responses, the LLM-based approach provides a lightweight and scalable alternative. By integrating natural-language-driven decision logic, this model circumvents the need for full system interoperability, making it highly adaptable to fragmented or heterogeneous supply chain environments.

8. Conclusions

In this study, distributed manufacturing was investigated using the Shared Warehouse game, simulating four scenarios to evaluate the effectiveness of AI in decision-making

for individual warehouse locations that represent autonomous production units within a broader concept of distributed manufacturing.

The results showed that AI is able to make rational and reasonable decisions even when constrained by random environmental factors. In scenarios without a bidding pool, the AI's decision-making resembled solving a mathematical problem similar to the secretary problem. Although a certain degree of randomness influenced the results, the AI consistently demonstrated an understanding of the consequences of its actions and tried to maximize the outcomes. However, AI's performance decreased in the task of managing multiple warehouse locations simultaneously, highlighting the efficiency of autonomous decision-making at the individual unit level. In scenarios with a bidding pool, the additional system complexity reduced randomness, resulting in better performance. The AI demonstrated a strong understanding of competitive dynamics and game theory, often rejecting suboptimal decisions that would unintentionally favor competitors. However, occasional inconsistencies in decision-making highlighted the areas where autonomous decision-making algorithms need further refinement.

Crucially, these results validate our proof-of-concept claim: LLM-driven agents outperformed random baselines on key metrics such as storage utilization, offer acceptance rates, and overall profit. By effectively leveraging capacity pooling and dynamic pricing strategies, the AI not only demonstrated technical feasibility but also tangible benefits in a competitive, decentralized environment. The successful outcomes highlight the practical advantages of integrating LLM agents into the decision-making process inside distributed manufacturing systems.

These results confirm the potential for delegating the decision-making tasks of autonomous production units to AI and demonstrate the feasibility of distributed production within the sharing economy. While AI can develop strategies to increase success in competitive environments, its performance improves when randomness is minimized. Nevertheless, some inconsistencies in recurring problems were found, suggesting that further development is needed to improve the robustness of AI systems.

Although several potentially efficient approaches exist for decentralized warehousing and capacity pooling, this study was positioned primarily as a proof of concept to demonstrate the feasibility and potential benefits of integrating LLM-driven agents. Hence, formal benchmarking against alternative methods was beyond our current scope. Nevertheless, future work should explore variations of the problem in distributed manufacturing, including head-to-head comparisons with other established strategies. In particular, examining AI performance in scenarios where other players employ derived (non-random) strategies, integrating consumer decision logic to reduce service costs, and creating a more comprehensive virtual environment with additional variables would yield deeper insights. Such advances can not only broaden our understanding of the system's competitiveness but also pave the way for more efficient and scalable AI-driven solutions in distributed manufacturing.

Author Contributions: Conceptualization, T.B.; Methodology, T.B., M.C. and S.V.; Software, M.C. and S.V.; Validation, T.B. and S.V.; Formal analysis, T.B. and M.C.; Investigation, M.C. and S.V.; Resources, M.C.; Data curation, M.C.; Writing—original draft, T.B. and M.C.; Writing—review & editing, T.B. and P.P.; Visualization, T.B.; Supervision, P.P.; Project administration, T.B. and P.P.; Funding acquisition, P.P. All authors have read and agreed to the published version of the manuscript.

Funding: Slovenian Research Agency: P2-0270; Ministry of Higher Education, Science and Technology of the Republic of Slovenia: 100-15-0510.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are openly available under GNU General Public License v3.0 in the project's GitHub repository at [<https://github.com/fsprojekti/shr-mfg-cpl-sim> (accessed on 11 December 2024)], and can be accessed using the reference [31].

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

References

1. Arioli, V.; Pezzotta, G.; Romero, D.; Adrodegari, F.; Sala, R.; Rapaccini, M.; Saccani, N.; Marjanovic, U.; Rakic, S.; West, S.; et al. Digital servitization business typologies in the manufacturing sector. *Int. J. Ind. Eng. Manag.* **2025**, *16*, 1–23. [[CrossRef](#)]
2. Marjanovic, U.; Lalic, B.; Medic, N.; Prester, J.; Palcic, I. Servitization in manufacturing: Role of antecedents and firm characteristics. *Int. J. Ind. Eng. Manag.* **2020**, *11*, 133–143. [[CrossRef](#)]
3. Slavic, D.; Marjanovic, U.; Medic, N.; Simeunovic, N.; Rakic, S. The Evaluation of Industry 5.0 Concepts: Social Network Analysis Approach. *Appl. Sci.* **2024**, *14*, 1291. [[CrossRef](#)]
4. Schmidtke, N.; Rettmann, A.; Behrendt, F. Matrix production systems-requirements and influences on logistics planning for decentralized production structures. In Proceedings of the 54th Hawaii International Conference on System Sciences, Kauai, HI, USA, 5 January 2021.
5. Guajardo, M.; Rönnqvist, M.; Flisberg, P.; Frisk, M. Collaborative transportation with overlapping coalitions. *Eur. J. Oper. Res.* **2018**, *271*, 238–249. [[CrossRef](#)]
6. Aliche, K.; Herrmann, J.; Weidmann, M. *Warehouses: The Boxex Worth €300 Billion*; Operations Extranet; McKinsey and Company: Chicago, IL, USA, 2018.
7. Pan, S. Horizontal Collaboration for Sustainable Transport and Logistics. Ph.D. Thesis, Université de Valenciennes et du Hainaut-Cambrésis, Valenciennes, France, 2017.
8. Milewski, D. Total costs of centralized and decentralized inventory strategies—Including external costs. *Sustainability* **2020**, *12*, 9346. [[CrossRef](#)]
9. Elia, V.; Gnoni, M.G.; Tornese, F. On-Demand Warehousing Platforms: Evolution and Trend Analysis of an Industrial Sharing Economy Model. *Logistics* **2024**, *8*, 93. [[CrossRef](#)]
10. Jamili, N.; Van Den Berg, P.L.; De Koster, R. Quantifying the impact of sharing resources in a collaborative warehouse. *Eur. J. Oper. Res.* **2022**, *302*, 518–529. [[CrossRef](#)]
11. Ferguson, T.S. Who solved the secretary problem? *Stat. Sci.* **1989**, *4*, 282–289. [[CrossRef](#)]
12. Ceschia, S.; Gansterer, M.; Mancini, S.; Meneghetti, A. Solving the online on-demand warehousing problem. *Comput. Oper. Res.* **2024**, *152*, 107011. [[CrossRef](#)]
13. Shahroudnejad, A.; Mousavi, P.; Perepelytsia, O.; Sahir; Staszak, D.; Taylor, M.E.; Bawel, B. A novel framework for automated warehouse layout generation. *Front. Artif. Intell.* **2024**, *7*, 1465186. [[CrossRef](#)] [[PubMed](#)]
14. Zhang, L.; Kumar, R. Framework for large language model applications in manufacturing systems. *Comput. Ind.* **2024**, *152*, 103608.
15. Hong, R.; Pang, X.; Zhang, C. Advances in Reasoning by Prompting Large Language Models: A Survey. *Cybern. Intell.* **2024**, 1–15. [[CrossRef](#)]
16. Rajendiran, G.R. Leveraging Large Language Models to Automate SOP in Warehouses Managed by WMS. *Int. J. Future Gener. Commun. Netw.* **2023**, *5*, 5857–5866.
17. Naim, M.M.; Potter, A.T.; Mason, R.J.; Bateman, N. The role of transport flexibility in logistics provision. *Int. J. Logist. Manag.* **2006**, *17*, 297–311. [[CrossRef](#)]
18. Wooldridge, M. *An Introduction to Multiagent Systems*; John Wiley & Sons: Hoboken, NJ, USA, 2009.
19. Wooldridge, M.; Jennings, N.R. Intelligent agents: Theory and practice. *Knowl. Eng. Rev.* **1995**, *10*, 115–152. [[CrossRef](#)]
20. Raiffa, H.; Schlaifer, R. *Applied Statistical Decision Theory*; John Wiley & Sons: Hoboken, NJ, USA, 2000; Volume 78.
21. Freeman, P. The secretary problem and its extensions: A review. In *International Statistical Review/Revue Internationale de Statistique*; John Wiley & Sons: Hoboken, NJ, USA, 1983; pp. 189–206.
22. Osborne, M.J. *A Course in Game Theory*; MIT Press: Cambridge, MA, USA, 1994.
23. Russell, P.N.; Norvig, P. *Artificial Intelligence: A Modern Approach by Stuart*; University of California, Berkeley: Berkeley, CA, USA, 2010.
24. Yu, C.; Xu, X.; Yu, S.; Sang, Z.; Yang, C.; Jiang, X. Shared manufacturing in the sharing economy: Concept, definition and service operations. *Comput. Ind. Eng.* **2020**, *146*, 106602. [[CrossRef](#)]
25. Vaswani, A. Attention is all you need. In Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 12 June 2017.
26. Bender, E.M.; Gebru, T.; McMillan-Major, A.; Shmitchell, S. On the dangers of stochastic parrots: Can language models be too big? In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual, 3–10 March 2021; pp. 610–623.
27. Chang, X.; Zhao, L. Collaborative strategies in the product-sharing market considering manufacturer's capital entry. *Int. J. Prod. Econ.* **2024**, *275*, 109345. [[CrossRef](#)]

28. Tong, H.; Zhu, J. A two-stage method for large-scale manufacturing service stable matching under uncertain environments in Cloud Manufacturing. *Comput. Ind. Eng.* **2022**, *171*, 108391. [[CrossRef](#)]
29. Rožman, N.; Diaci, J.; Corn, M. Scalable framework for blockchain-based shared manufacturing. *Robot.-Comput.-Integr. Manuf.* **2021**, *71*, 102139. [[CrossRef](#)]
30. Wang, W.; Yue, S. An inventory pooling model for spare units of critical systems that serve multi-companies. *Transp. Res. Part E Logist. Transp. Rev.* **2015**, *76*, 34–44. [[CrossRef](#)]
31. Corn, M. SHR-MFG-CPL-SIM. Available online: <https://github.com/fsprojekti/shr-mfg-cpl-sim.git> (accessed on 11 December 2024).

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