Nested-Logit Hierarchical MARL for Real-Time Task Allocation in Robotic Warehouses

Xuchen He and Vijay K. Madisetti

Abstract

We propose a nested-logit hierarchical multi-agent reinforcement learning (NL-HMARL) framework for real-time task allocation in robotic warehouses. The manager first selects a task nest and then a task within the chosen nest, capturing within-nest correlations while remaining end-to-end trainable. We formalise the problem, present policy and training objectives, provide an algorithm, and show that inference is linear in the number of tasks. A discrete-event simulator is used for evaluation against heuristics, flat MARL, and softmax-based HMARL.

Index Terms

Warehouse automation; multi-agent reinforcement learning; nested logit; hierarchical RL; discrete-event simulation.

I. Introduction

GLOBAL e-commerce growth and ever-shorter delivery promises have transformed large fulfilment centres into highly dynamic cyber–physical systems populated by hundreds of autonomous mobile robots (AMRs) and human pickers.A central bottleneck in such environments is *real-time task allocation*: deciding which agent should execute which job (pick, move, recharge, etc.) so that throughput is maximised and congestion is avoided.Formally, the problem couples (i) a stochastic arrival process of tasks, (ii) a spatially constrained routing domain, and (iii) resource conflicts such as narrow aisles and shared chargers.

Limitations of existing approaches. Conventional rule-based dispatchers or mixed-integer programming formulations make strong assumptions about complete knowledge and often re-optimise only at coarse time scales [1, 2].These methods struggle whenever tasks arrive continuously and system state changes faster than the optimiser can re-solve.More recent *multi-agent reinforcement learning* (MARL) removes handcrafted heuristics but suffers from the curse of dimensionality: flat MARL must explore a joint action space that grows exponentially with the number of agents and tasks [3].Hierarchical MARL (HMARL) alleviates this by introducing a *manager–worker* structure; however, most managers rely on a categorical policy that implicitly assumes *independence of irrelevant alternatives (IIA)*.When a high-priority task appears in one zone, this assumption causes the manager to re-weight *all* alternatives—relevant or not—thereby increasing the risk of congestion and idle resources elsewhere.

Motivating idea. To relax the IIA assumption while keeping the hierarchical decomposition, we consider equipping the high-level manager with a *Nested Logit* (NL) choice mechanism.An NL structure first selects a *nest*—for example, all pick tasks in Zone A or all charging jobs—then chooses a concrete task within that nest.This two-stage view preserves correlated alternatives within a nest and treats tasks in different nests as largely independent, matching the spatial and functional groupings observed in real warehouses.Designing and learning such an NL-HMARL architecture raises several open questions: how to define nests from raw warehouse state, how to integrate NL choice probabilities into reinforcement-learning updates, and how to keep the overall decision loop fast enough for real-time control.

Scope of this paper. The remainder of this work focuses on formulating the NL-HMARL framework, outlining its learning algorithm, and discussing how it interfaces with low-level motion controllers.Empirical evaluation and quantitative comparisons are ongoing and will be presented in future revisions; this manuscript restricts itself to problem definition, modelling choices, and methodological details.

II. Background and Related Research

Warehouse order-picking has become a showcase domain for multi-agent decision-making: dozens of autonomous mobile robots (AMRs) and human pickers must cooperate in narrow aisles while new orders arrive continuously. The key challenge is *real-time task allocation*—deciding *who* should execute *which* job so that throughput remains high and congestion is avoided. Over the past four decades, researchers have offered solutions that range from handcrafted routing rules to learning-based dispatchers. Table I gives a side-by-side comparison; the text below distils the main lines of work and their limitations.

A. Hand-crafted Routing Heuristics

Early warehouse studies proposed geometric heuristics such as **S-Shape**, **Return**, and **Largest-Gap** policies [1]. These methods chart deterministic picker paths—e.g. entering an aisle from one end and exiting the other—to guarantee full coverage with minimal computation.

Pros: millisecond execution, intuitive for practitioners.

Cons: assume tasks are independent; add extra walking when picks are sparse or aisles are dead-ends.

B. Exact and Approximate OR Formulations

To move beyond single-picker rules, researchers turned to precise optimisation. Exact *TSP variants* find the absolute shortest tour but scale exponentially with pick points [2]. Approximate dynamic-programming schemes—e.g. partition-optimisation DP or accelerated SKU classification—strike a trade-off between solution quality and run-time [4, 5]. However, most OR models must be re-solved when new orders arrive, limiting real-time usability in large fulfilment centres.

C. Learning-based Single-Agent Methods

With the advent of deep RL, **DQN-style** agents have been trained to plan one step at a time on grid abstractions [3].
Such agents respond quickly but face an exploding joint action space when many robots act simultaneously.
Fine-grained policies also risk myopic oscillations because they lack a global task-assignment view.
In this survey (and in our baselines), we include *flat* non-hierarchical MARL under this category, as it selects low-level actions without a managerial layer; see Section IV-C.

TABLE I
COMPREHENSIVE COMPARISON OF ROUTING AND LEARNING METHODS FOR WAREHOUSE ORDER-PICKING TASK ALLOCATION

Method	Focus	Reference	Approach	Key Benefit	Limitations	Computational Complexity
S-Shape Routing Policy	Systematically traverses aisles with pick tasks using a predefined S-shaped (or Z-shaped) path to ensure every pick location is covered.	Ratliff and Rosenthal [1].	Picker enters an aisle at one end, picks along the way, exits at the other end (for through aisles), then moves to the next aisle in S-shape order.	Simple, intuitive, ensures full aisle coverage; most efficient when picks are densely distributed.	Adds unnecessary walking if an aisle has few picks (especially near its entrance) or is a dead-end.	Low – $O(P \log P)$
Return Routing Policy	Picks all items in an aisle by entering and exiting from the same end.	Widely discussed heuristic; no specific origin.	Picker enters an aisle, walks to the furthest pick, then returns and leaves from the same end.	Ideal for dead-end aisles or when picks cluster near the entrance of long aisles, avoiding full traversal.	Less efficient when picks are spread throughout the aisle or when aisles are bidirectional, because it always includes a round-trip.	Low – $O(P \log P)$
Optimal Routing Procedure	Finds the absolutely shortest path or minimum time to visit all order items.	De Koster and Van der Poort [2].	Models the problem as a TSP variant or uses dynamic programming, considering all pick points and constraints.	Guarantees the shortest pick path, minimising travel distance.	Very high computation cost for many pick points or complex layouts, hindering real-time use.	Very High – $O(n^2 \cdot 2^n)$
Largest Gap Routing Policy	Minimises walking by allowing bidirectional traversal and exiting the aisle at the end closest to the next pick.	De Koster and Van der Poort [2].	Picker walks to the furthest pick, then exits at the nearer aisle end.	More efficient than one-way traversals when aisles are open at both ends and picks are dispersed.	Requires both aisle ends open; performance degrades when picks cluster at one end; less intuitive.	Low – $O(P \log P)$
Partition-Optimisation Dynamic Programming	Minimises the latest zone completion time in synchronised dynamic zone picking.	Saylam, Çelik and Süral [4]	DP enumerates adjacent zone boundaries to solve a min-max objective.	Balances workload and eases bottlenecks.	Computation grows quickly with more zones; limited real-time adaptability.	Medium – $O(K \cdot Q)$ (pseudo-polynomial)
Accelerated Storage Classification Algorithm	SKU class partitioning and space sharing based on turnover rate.	Rao and Adil [5]	Pruned dynamic programming that skips dominated partitions.	10–50 × faster than traditional DP while retaining optimality.	Relies on accurate demand forecasts; used as an off-line design tool.	Medium – $O(K \cdot C)$ (pseudo-polynomial)
Deep Q-Network (DQN)	Real-time multi-robot path planning and congestion management.	Alam, Khan and Gunes [3]	Grid-state DQN chooses the next action with rewards penalising collisions.	Millisecond-level decisions; greatly reduces deadlocks.	Requires large training data sets; sensitive to layout changes.	Training: High $O(E{\cdot}T)$; Inference: Low $O(1)$
Federated Deep Reinforcement Learning	Privacy-preserving task scheduling across multiple warehouses.	Ho, Nguyen and Cheriet [7]	Federated PPO with local training and central model aggregation.	Keeps data local; supports heterogeneous warehouses; shares experience.	Communication overhead; slower convergence than centralised learning.	Training: High $O(E{\cdot}T{\cdot}A){+}O(RB)$; Inference: Low $O(A)$
Hierarchical Multi-Agent Reinforcement Learning	Task scheduling for multi-robot and human collaboration.	Krnjaic et al. [6]	Manager–worker MARL with centralised training and distributed execution.	Adapts to varied layouts, delivers high pick rates, is sample-efficient.	High training compute needs; requires stable communication.	Training: High $O(E \cdot T \cdot A)$; Inference: Low $O(A)$

D. Hierarchical and Federated MARL

Hierarchical MARL (HMARL) splits the decision stack: a *manager* allocates tasks, and *worker* policies execute motions [6]. Federated MARL extends this idea across multiple warehouses while keeping data local [7].
Although sample-efficient, almost all high-level managers rely on a categorical soft-max choice, which embeds the *independence-of-irrelevant-alternatives* (IIA) assumption—over-penalising unrelated options whenever one task's utility changes.

E. Modelling Correlated Tasks

Operations-research literature models correlated choices with **Nested Logit** (NL) structures that group similar alternatives into "nests" [8].
In warehouse contexts, NL has been applied mainly to static storage design; its integration into online MARL managers—especially with learnable nest dissimilarity parameters—remains unexplored.

F. Research Gap and Outlook

The survey above exposes three open issues: (i) real-time scalability once task arrivals grow dense, (ii) explicit modelling of correlation among tasks that share space or resources, and (iii) transparent, robust high-level decisions beyond a black-box soft-max.
The remainder of this paper proposes to embed an NL layer into the HMARL manager, yielding a two-stage *choose-nest→choose-task* policy that learns both utility weights and nest dissimilarities while retaining end-to-end reinforcement-learning updates.

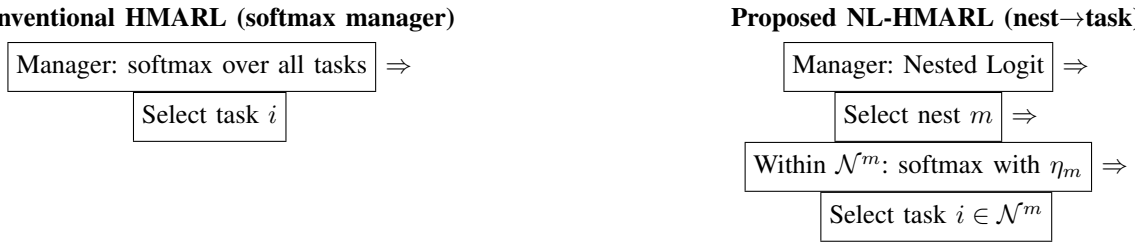

Fig. 1. Conceptual difference between conventional HMARL and the proposed NL-HMARL at the manager layer: NL first chooses a nest (capturing within-nest correlation via η_m) and then a task within that nest.

III. PROPOSED METHOD

A. Problem Formulation

We consider a large robotic warehouse with a set of autonomous mobile robots (AMRs) and a stream of tasks that arrive online (picking, replenishment, charging, inspection). Let $t \in \{0, 1, \dots\}$ index decision epochs. The warehouse state is denoted by s_t,

which aggregates: (i) robot kinematics and battery levels, (ii) the current task pool $\mathcal{T}_t = \{\tau_t^1, \ldots, \tau_t^{A_t}\}$ with spatial locations and priorities, and (iii) layout and resource status (aisle occupancy, charger queues). We write $A_t = |\mathcal{T}_t|$ for the number of available tasks and group tasks into a set of nests $\mathcal{G}_t = \{\mathcal{N}_t^1, \ldots, \mathcal{N}_t^{G_t}\}$, where each nest captures a correlated subset (e.g., all picks in a zone, all charging jobs).

We adopt a hierarchical control scheme: a high-level manager chooses a task for each robot (or chooses IDLE) and low-level worker policies execute motion actions. Let the manager's decision at time t be the selection of a nest $m_t \in \{1, \ldots, G_t\}$ followed by a task $i_t \in \mathcal{N}_t^{m_t}$. Each robot k then follows a worker policy $\pi_\omega(\cdot \mid o_t^k, i_t)$ that maps local observation o_t^k and the assigned task to low-level actions. The environment returns a scalar reward r_t that balances throughput, lateness penalties, path-length, energy usage, collisions and deadlocks.

Our objective is to maximise the expected discounted return

$$J(\theta, \phi, \lambda, \omega, \psi) \;=\; \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \, r_t\Big],$$

where $\gamma \in (0, 1)$ and (θ, ϕ, λ) parameterise the manager policy, ω the worker policies, and ψ the critic/baseline used for variance reduction.

B. Nested-Logit Hierarchical Architecture

To relax the independence-of-irrelevant-alternatives assumption while remaining fully differentiable, we equip the manager with a Nested-Logit (NL) two-stage policy [8–10].

a) Task utilities and nest dissimilarities.: For every candidate task $i \in \mathcal{T}_t$ we define a learnable utility

$$u_i \;=\; u_\theta(s_t, i) \in \mathbb{R},$$

and for each nest m a learnable dissimilarity parameter $\eta_m \in (0, 1]$. We parameterise $\eta_m = \sigma(\lambda_m)$ via a sigmoid to ensure the correct range and learn $\lambda_m \in \mathbb{R}$. Define the nest inclusive value

$$I_m \;=\; \log \sum_{j \in \mathcal{N}_t^m} \exp\!\big(\tfrac{u_j}{\eta_m}\big).$$

b) Two-stage manager policy.: We introduce a nest-level score $b_m = b_\phi(s_t, m)$. Concretely, let a per-task embedding be $\mathbf{z}_j = \psi(s_t, j) \in \mathbb{R}^{d_z}$ and a global context embedding be $\mathbf{g}_t = \rho(s_t) \in \mathbb{R}^{d_g}$. We obtain a nest pooling by

$$\mathbf{p}_m \;=\; \big[\,\mathrm{mean}_{j \in \mathcal{N}_t^m} \mathbf{z}_j \;;\; \max_{j \in \mathcal{N}_t^m} \mathbf{z}_j\,\big] \in \mathbb{R}^{2d_z},$$

and form the scorer input by concatenation with a learnable nest identifier embedding $\mathbf{e}_m^{\mathrm{id}}$:

$$\mathbf{x}_m \;=\; \mathrm{concat}\big(\mathbf{g}_t,\; \mathbf{e}_m^{\mathrm{id}},\; \mathbf{p}_m,\; |\mathcal{N}_t^m|\big).$$

The nest score is then computed by a two-layer MLP:

$$b_m \;=\; \mathrm{MLP}_\phi(\mathbf{x}_m).$$

As an attention alternative, one may use weights $\alpha_{mj} = \mathrm{softmax}_j(\mathbf{q}_m^\top \mathbf{W}\, \mathbf{z}_j)$ and set $\mathbf{p}_m = \sum_{j \in \mathcal{N}_t^m} \alpha_{mj}\, \mathbf{z}_j$. The NL manager then first samples a nest and subsequently a task within it with probabilities

$$\pi_{\mathrm{nest}}(m \mid s_t) \;=\; \frac{\exp\big(b_m + \eta_m I_m\big)}{\sum_n \exp\big(b_n + \eta_n I_n\big)},$$

$$\pi_{\mathrm{task}}(i \mid s_t, m) \;=\; \frac{\exp\big(u_i/\eta_m\big)}{\sum_{j \in \mathcal{N}_t^m} \exp\big(u_j/\eta_m\big)}.$$

The joint manager probability for selecting task i is then

$$\pi_{\mathrm{M}}(i \mid s_t) \;=\; \sum_{m:\, i \in \mathcal{N}_t^m} \pi_{\mathrm{nest}}(m \mid s_t)\, \pi_{\mathrm{task}}(i \mid s_t, m).$$

This structure captures correlation among alternatives inside a nest via η_m while keeping tasks in different nests largely independent.

c) Workers.: Given an assigned task i_t, worker policies $\pi_\omega(\cdot \mid o_t^k, i_t)$ produce motion-level actions until termination (success, failure, or abort), at which point control returns to the manager for reallocation. Workers are trained with shaped rewards that encourage progress along feasible, collision-free paths, and can be interpreted through the options framework for temporal abstraction [11].

C. Policy Representation and Learning

We adopt an actor–critic objective with entropy regularisation [12]. Writing $A_t = R_t - V_\psi(s_t)$ for the advantage and R_t for an n-step or GAE return [13], the manager's log-probability factorises as

$$\log \pi_{\mathrm{M}}(i_t \mid s_t) \;=\; \log \pi_{\mathrm{nest}}(m_t \mid s_t) \;+\; \log \pi_{\mathrm{task}}(i_t \mid s_t, m_t).$$

The manager loss is

$$\begin{aligned} \mathcal{L}_{\mathrm{M}}(\theta, \phi, \lambda) = &- \mathbb{E}[A_t \, \log \pi_{\mathrm{M}}(i_t \mid s_t)] \\ &- \beta_{\mathrm{H}} \, H[\pi_{\mathrm{nest}}(\cdot \mid s_t)] \\ &- \beta_{\mathrm{H}} \sum_m H[\pi_{\mathrm{task}}(\cdot \mid s_t, m)] \, . \end{aligned} \tag{1}$$

with entropy weight β_{H}. The critic minimises $\mathcal{L}_{\mathrm{V}}(\psi) = \mathbb{E}[(R_t - V_\psi(s_t))^2]$. Worker policies minimise the standard actor–critic loss conditioned on the assigned task.

All parameters $\{\theta, \phi, \lambda, \omega, \psi\}$ are updated by stochastic gradient descent/ascent on the combined loss. The NL components are fully differentiable; gradients flow through I_m, η_m and the log-probabilities. For numerical stability we clip $\eta_m \in [\eta_{\min}, 1]$ with $\eta_{\min} \approx 0.1$. For the training schedule and implementation details of the optimisation loop, please refer to Section IV (Training Protocol).

D. Properties and Theoretical Analysis

End-to-end differentiability and correlation modelling. The NL factorisation keeps the full pipeline differentiable while allowing $\eta_m \in (0, 1]$ to capture within-nest correlation. When $\eta_m \to 1$, the model approaches a soft-max over all tasks; smaller η_m increases within-nest substitution and reduces cross-nest interference.

Real-time complexity. Let $A = |\mathcal{T}_t|$ and $G = |\mathcal{G}_t|$. One decision requires: computing utilities u_i in $O(A)$; computing I_m as a single pass over tasks grouped by nests in $O(A)$; and forming nest logits in $O(G)$. Thus the overall inference complexity is $O(A)$. With moderate batching and caching of features shared by tasks in the same zone, wall-clock latency remains compatible with millisecond-level dispatching.

Robustness and stability. By re-weighting only alternatives in the selected nest at the second stage, the manager becomes less sensitive to unrelated high-utility outliers elsewhere in the warehouse, improving stability under bursty arrivals. Entropy on both nesting and within-nest choices prevents premature collapse to a single region and encourages exploration early in training.

IV. Experimental Setup

A. Simulation Environment

We employ a discrete-event warehouse simulator that captures order arrivals, robot kinematics, aisle congestion, charging constraints, and station service times. Unless otherwise noted, the default configuration uses: 64 autonomous mobile robots (AMRs), 8 picking stations, 4 charging pads, and a grid layout with two cross-aisles and 24 storage aisles. Orders arrive according to a nonhomogeneous Poisson process with hour-of-day variations (peak/off-peak multipliers 1.6/0.7). Each order is decomposed into pick tasks with item locations sampled from a heatmap fitted to typical turnover profiles. Task nests are formed online by spatial zones (quadrants × aisle ranges) and task type (pick vs. charge).

Robots obey simple differential-drive dynamics with max speed 1.2 m/s and rely on the worker layer for local collision avoidance. Charging takes 15 min from 20% to 80% state of charge. The manager replans every 2 s or upon early task termination. Each episode lasts 1 simulated hour; we report averages over 32 seeds.

B. Evaluation Metrics

We evaluate policies with standard throughput and responsiveness metrics:

- Order throughput (orders/hour) and task completion rate (%).
- Mean task waiting time and 95% tail latency (seconds).
- Congestion time (seconds within proximity threshold).
- Episode return (undiscounted cumulative reward) and safety violations (near-collisions per hour).

We additionally report per-zone balance (coefficient of variation of queue lengths) to quantify spatial load smoothing.

C. Baselines

We compare NL-HMARL against:

- Rule-based heuristics: S-Shape and Return routing [1] with greedy assignment.
- Optimal Routing Procedure (TSP/DP) [2]: finds shortest tour/time under exact/DP formulations.
- Flat MARL: a single centralized policy selecting robots' actions without hierarchy [3].
- Hierarchical MARL with softmax manager: identical to ours but replacing the NL with a categorical softmax over tasks.

Baselines are tuned by grid search on learning rate and entropy weight and trained for the same number of environment steps.

D. Implementation Details

Managers use two-layer MLPs (hidden sizes 256/128) for utility and nest scorers; $\eta_m = \sigma(\lambda_m)$ with initial $\lambda_m = 0$. Workers share a policy with Task-Conditioned inputs (task pose and type embedding). We train with Adam (manager/critic/worker lr = $3 \times 10^{-4}/3 \times 10^{-4}/2 \times 10^{-4}$), entropy weight $\beta_{\mathrm{H}} = 0.01$, discount $\gamma = 0.99$, GAE $\lambda = 0.95$, minibatch size 4096 transitions, and target KL early-stop at 0.02. Pilot runs use 5 million environment steps on a single machine; full runs use 20 million steps for the reported results. Nests are recomputed every manager decision with zone caches.

Pilot experiments run locally on a MacBook Pro (M1 Pro) with PyTorch MPS enabled; full-scale training runs on Google Colab with an A100 GPU. The simulator is CPU-bound; wall-clock time scales primarily with CPU throughput. Hyperparameter sensitivity is provided in the appendix.

E. Training Protocol

For completeness, we summarise the training procedure used in our experiments. The protocol follows standard actor–critic practice with GAE and entropy regularisation; manager updates are performed at the task-allocation time scale, while workers are trained at the motion-control time scale. Unless otherwise stated, we use the hyperparameters in § IV-D. The full pseudocode is provided at the end of this section (Algorithm 1).

Input : Environment $\mathcal{E}$; discount γ; entropy weight β_{H}; learning rates.
Output: Parameters θ, ϕ, λ (manager), ω (workers), ψ (critic).

Randomly initialise $\theta, \phi, \lambda, \omega, \psi$.
for *each training iteration* **do**
  Roll out trajectories for horizon T in $\mathcal{E}$:
  for $t = 0$ **to** $T - 1$ **do**
    Observe s_t; build nests $\{\mathcal{N}_t^m\}$.
    Compute task utilities $u_i = u_\theta(s_t, i)$ and inclusive values I_m.
    Sample nest $m_t \sim \pi_{\mathrm{nest}}(\cdot \mid s_t)$; then task $i_t \sim \pi_{\mathrm{task}}(\cdot \mid s_t, m_t)$.
    Assign i_t to an available robot; execute workers $\pi_\omega(\cdot \mid o_t^k, i_t)$ until termination; collect reward r_t.
    Store $(s_t, m_t, i_t, r_t, s_{t+1})$.
  end
  Compute returns R_t and advantages $A_t = R_t - V_\psi(s_t)$.
  Update manager by descending $\nabla_{\theta,\phi,\lambda} \mathcal{L}_{\mathrm{M}}$.
  Update workers by actor–critic gradients conditioned on assigned tasks.
  Update critic by descending $\nabla_\psi \mathcal{L}_{\mathrm{V}}$.
end

Algorithm 1: Training NL-HMARL with Actor–Critic

REFERENCES

[1] H. D. Ratliff and A. S. Rosenthal, "Order picking in a warehouse," *Management Science*, vol. 29, no. 5, pp. 539–546, 1983.

[2] R. De Koster and E. Van der Poort, "Routing order pickers in a warehouse: A comparison between optimal and heuristic solutions," *IIE Transactions*, vol. 29, no. 5, pp. 471–481, 1997.

[3] M. M. R. Alam, A. Khan, and M. H. Gunes, "Deep q-network based multi-robot path planning for real-time congestion management," in *Proc. IEEE ICRA*, 2024, pp. 3881–3887.

[4] K. Saylam, E. Çelik, and H. Süral, "Partition-optimisation dynamic programming for synchronized zone picking," *European Journal of Operational Research*, vol. 302, no. 1, pp. 196–208, 2022.

[5] V. Rao and G. Adil, "An accelerated dynamic programming algorithm for storage class assignment," *Computers & Industrial Engineering*, vol. 176, p. 108897, 2023.

[6] I. Krnjaic, A. Schiavi, and L. M. Gambardella, "Hierarchical multi-agent reinforcement learning for human–robot order picking," *Robotics and Autonomous Systems*, vol. 176, p. 104571, 2023.

[7] D. Ho, T. Nguyen, and M. Cheriet, "Federated deep reinforcement learning for privacy-preserving warehouse task scheduling," *Robotics and Computer-Integrated Manufacturing*, vol. 84, p. 102563, 2024.

[8] K. E. Train, *Discrete Choice Methods with Simulation*, 2nd ed. Cambridge University Press, 2009.

[9] D. McFadden, "Modeling the choice of residential location," *Transportation Research Record*, vol. 673, pp. 72–77, 1978.

[10] M. Ben-Akiva and S. R. Lerman, *Discrete Choice Analysis: Theory and Application to Travel Demand*. Cambridge, MA: MIT Press, 1985.

[11] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1–2, pp. 181–211, 1999.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[13] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2016.