Appears in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Hangzhou, China. 2025

Learning to Solve the Multi-Agent Task Assignment Problem for Automated Data Centers

Christelle Loiodice, Sofia Michel, Darko Drakulic, Jean-Marc Andreoli
NAVER LABS Europe
https://github.com/naver

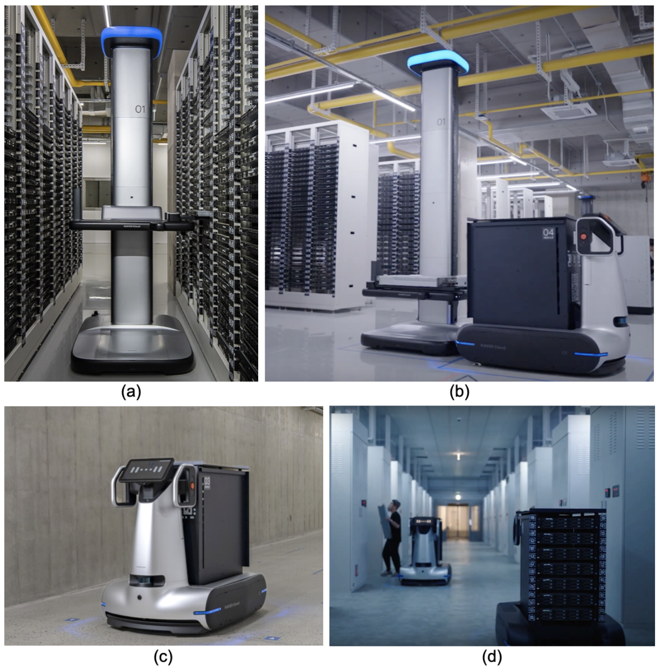

Figure 1: Illustration of a workflow in an automated data center: (a) a warehouse robot retrieves a server from storage; (b) then transfers it on a delivery robot at a workstation; (c) the delivery robot transports the server to the designated server room; (d) it meets a human worker who retrieves and installs the server. Possibly the delivery robot goes back to the warehouse with faulty parts, where a warehouse robot retrieves the assets and places them in storage, creating a second sequence of steps (c), (b) and (a). Photos: © NAVER CORP. ALL RIGHTS RESERVED.

Abstract

We consider a large-scale data center where a fleet of heterogeneous mobile robots and human workers collaborate to handle various installation and maintenance tasks. We focus on the underlying multi-agent task assignment problem which is crucial to optimize the overall system. We formalize the problem as a Markov Decision Process and propose an end-to-end learning approach to solve it. We demonstrate the effectiveness of our approach in simulation with realistic data and in the presence of uncertainty.

1. Introduction

With the increasing demand for hyperscale AI, automated data centers must not only scale their computing power and data storage but also improve operational efficiency. This need has driven the adoption of robotic automation for handling various tasks such as installing and maintaining computing assets. However, efficient coordination of robotic fleets remains a significant challenge, particularly when multiple heterogeneous agents, both mobile robots and human workers, must collaborate to execute complex tasks under time constraints.

In this work, we consider a large data center where a fleet of heterogeneous agents, including warehouse robots, delivery robots, and human workers, collaborate to fulfill installation and maintenance tickets.

A typical ticket consists of retrieving assets (servers or small parts) from a warehouse, transporting them, and processing them at a designated server rack, as illustrated in Figure 1. This workflow exhibits several *collaborative tasks* with some dependency constraints: (i) a warehouse robot retrieves the assets and transfers them to a delivery robot at an intermediate workstation; (ii) a human worker meets the delivery robot at the server rack for installation or maintenance; and possibly (iii) the delivery robot goes back to the warehouse loaded with faulty parts which are then retrieved by a warehouse robot and placed in storage. When scheduling those tasks, one must take into account their time dependency and the fact that they should all involve the same delivery robot while the other agents are free to do other tasks in-between.

We formalize the optimization of such system as a Multi-Agent Task Assignment problem (MATAP), where a centralized planner assigns tasks to agents of the required types and produce optimized schedules, typically minimizing either the sum of task completion times or the makespan (maximum task completion time). This problem is strongly NP-hard, as it belongs to the family of single-task robots, multi-robot tasks, time-extended assignment problems in the seminal taxonomy [9], with the additional challenge of cross-schedule dependencies [15]. Due to this computational complexity, exact optimization methods fail to scale to real-world settings, making specialized heuristics a more suitable choice. However such heuristics heavily rely on expert knowledge and must be manually tailored to each specific use-case. Recently, machine learning approaches have emerged as a promising alternative to learn data-driven customized heuristics for NP-hard optimization problems without the need for domain-specific expertise [3, 5]. These approaches particularly excel in scenarios where similar optimization instances are solved repeatedly, allowing a model to learn structural patterns to produce a customized heuristic. However, existing ML-based heuristics have primarily been developed for classical NP-hard problems with relatively simple constraints such as vehicle routing [14, 28] or job shop scheduling problems [16,30], making their application to richer real-world problems such as the MATAP nontrivial.

In this paper, we propose an end-to-end learning framework to solve the MATAP. We start by formalizing a generic version of the MATAP as a combinatorial optimization problem and the construction of optimal solutions as a Markov Decision Process (MDP). To effectively capture the problem's features and constraints, we introduce a graph-based state representation and design a customized encoder and decoder for a transformer-based policy network. Then we explore multiple training strategies to learn the MDP policy, including imitation learning, using solutions generated by a simple greedy heuristic, and self-supervised learning. Finally, we validate our approach using a realistic simulator for a fleet of robots in a data center environment and demonstrate its effectiveness in dealing with various numbers of agents and tasks as well as environment uncertainties through re-planning. In summary, our key contributions are:

- **Problem Formalization**: we formalize the MATAP in data centers as a combinatorial optimization problem and design an efficient MDP to model the solution construction process.
- **Neural Network Design**: we adapt a transformer-based policy architecture to effectively encode the the MATAP's intricate state and action spaces.
- **Efficient Training**: we propose an efficient training strategy, combining a supervised pretraining followed by a self-supervised fine-tuning.
- **Empirical Evaluation**: we benchmark our approach in realistic simulator against classic scheduling heuristics, adapting them to our setting and show the superiority of our learned policies, across various combinations of tasks and agents distributions and in the presence of uncertainty.

2. Related Works

Multi-Agent Optimization in Automated Warehouses. Several works address multi-agents optimization in automated warehouses. For example, [17] considers a warehouse scenario where products need to be transported by mobile robots from shelves to packaging stations but without a specific coordination between the robots. [29] uses Large Neighborhood Search to optimize the task assignment for a multi-agent pickup and delivery scenario. Several works focus on multi-agent path finding, i.e. finding optimal collision-free paths for a fleet of robots, which is crucial in warehouses [19, 21]. In this paper, we focus on the task assignment and scheduling while the path planning and collision avoidance is handled by independent modules (either the robot's local controllers or a multi-robot path finding planner) which provide estimates of the travel times that used within the MATAP.

Multi-Robot Task Allocation (MRTA). There are many works addressing different variants of MRTA, using a range of techniques including integer programming, auction-based methods and graph-based methods, see surveys [10, 12, 26]. Learning-based approaches for multi-robot applications are surveyed in [22]. The closest works to ours are [27] and [24] which address the single-task single-robot setting, with in-schedule or cross-schedule dependencies respectively, using reinforcement or imitation learning. The only learning-based approach which addresses a multi-robot task setting is [23] but it considers the case where the multi-robot tasks do not require the explicit collaboration between different agents but rather that the sum of time steps spent on a given task by any of the homogeneous agents corresponds to the workload of the task. On the other hand, learning-based approaches have been successful for classical scheduling problems such as the (flexible) job shop scheduling problem [7,16,25, 30] but these scheduling problems are much simpler than the MATAP.

3. Problem description

3.1. MATAP description

In this section we introduce the data and notations and formulate the MATAP optimization problem.

Agents and Tasks. We consider a finite set of tasks T, and a finite set of heterogeneous agents A, partitioned according to skills into subsets of homogeneous agents: $A=\biguplus_{s\in S} A_s$. A task $u\in T$ happens at a certain location, it has a processing time δ_u and requires the collaboration of agents of different skills, expressed through a binary parameter λ_{su} which indicates whether task u requires an agent of skill s (if it does, we assume it requires exactly one agent of that skill; furthermore, we assume a task always requires at least one skill). For two tasks u, v, we denote by δ_{auv} the time it takes for agent a to be ready to start task v after finishing task u (which typically includes the travel time between the tasks locations).

Precedence Constraints. In our application, the workflow is split between several collaborative tasks which are interdependent and need to be executed in a certain order and involve the same agent of a certain skill. Typically a delivery robot loaded by a warehouse robot (task u) is the same one that should then meet a worker at a designated server rack (task v). We express this requirement as a *precedence constraint*, using a binary parameter λ_{suv} to indicate that task v must be executed right after task u by the same agent of skill s. Note that there are no constraints on the agents of the other skills required by u and v.

Initial Tasks. To allow the definition of MATAP in a dynamic setting where some agents may already be executing some tasks (see Sec. 3.3), we introduce the concept of *initial tasks*, which are not part of T and possibly dummy. We denote by $i(a)$ the initial task of agent a. Its processing time $\delta_{i(a)}$ corresponds to the time when agent a will be available to be assigned tasks in T. We denote by δ_{av} (short for $\delta_{ai(a)v}$) the time it would take agent a to be ready to start a task v after finishing $i(a)$. Finally to express possible precedence constraints linked to the initial tasks, we denote by λ_{av} a binary parameter that indicates if agent a must execute task v immediately after its initial task. We denote by T_0 the set of initial tasks and let $\bar{T}:=T\cup T_0$.

Plans and Schedules. The goal of MATAP is to compute an assignment of the tasks of T to the agents and an associated schedule such that all the constraints are satisfied and a certain objective function is optimized. The decision variables are *plans*, i.e. sequences of tasks for each agent. We denote by $\mathcal{X}$ the space of such plans. For a plan $x\in\mathcal{X}$, we denote by x_{auv} the binary indicator that agent a is assigned to task v immediately after task u in x. Given a plan, we can construct a valid schedule, where the start time y_v and completion time z_v of each task v are defined by:

$$\begin{aligned} y_v &:= \begin{cases} 0 & \forall v\in T_0, \\ \max\limits_{a\in A} \sum_{u\in\bar{T}} (z_u+\delta_{auv}) x_{auv} & \forall v\in T, \end{cases} \\ z_v &:= y_v+\delta_v \qquad\qquad\qquad\qquad \forall v\in\bar{T}. \end{aligned} \tag{1}$$

Objective Function. We denote the objective function as F and use in our experiments two standard metrics which are the sum of completion times $C_{\text{sum}}=\sum_{u\in\bar{T}} z_u$ or the makespan (maximal completion time): $C_{\max}=\max_{u\in\bar{T}} z_u$.

MATAP Formulation. We can now formulate our multi-agent task assignment problem, parametrized by the tuple $\langle A, T, \delta, \lambda\rangle$ as the following optimization problem:

$$\begin{array}{lllr} \min_x & F(z) & & \\ \text{s.t.} & z_u = y_u + \delta_u & \forall u\in\bar{T}, & (a) \\ & y_v \geq \sum_{u\in\bar{T}} (z_u+\delta_{auv}) x_{auv} & \forall v\in T, a\in A, & (b) \\ & \sum_{a\in A_s}\sum_{u\in\bar{T}} x_{auv} = \lambda_{sv} & \forall v\in T, s\in S, & (c) \\ & \sum_{a\in A_s} x_{auv} \geq \lambda_{suv} & \forall u,v\in T, s\in S, & (d) \\ & x_{ai(a)v} \geq \lambda_{av} & \forall a\in A, v\in T & (e) \\ & x\in\mathcal{X}. & & \end{array}$$

Constraints (a-b) reflect the completion and start time definitions (1); (c) expresses that the number of agents of skill s assigned to a task v is exactly $\lambda_{sv}\in\{0,1\}$; (d) ensures that if there is a precedence constraint between tasks u, v for a skill s ($\lambda_{suv}=1$), then at least one agent from that skill is assigned to v immediately after u; and similarly (e) ensures the precedence constraint satisfaction for the initial tasks. Note that y and z are intermediary variables and can be removed, since they are completely determined by the x variable by the definitions (1). This optimization problem is therefore a purely combinatorial optimization problem on the discrete space $\mathcal{X}$ of plans.

3.2. Formalization as an MDP

We formalize the construction of plans for the MATAP as a Markov Decision Process (MDP). We use the BQ-NCO approach [8] which allows the definition of an efficient MDP whose optimal policy produces optimal solutions for the original combinatorial optimization (CO) problem. The BQ approach relies on two assumptions: (i) the CO problem must satisfy the Bellman optimality principle of dynamic programming [2,4] and (ii) local conditions on the actions ensure the feasibility of the final solution. We will see that both conditions hold for MATAP. We now define each component of the MDP:

States. A state is an instance of MATAP, parametrized by a tuple $\langle A, T, \delta, \lambda\rangle$. The initial state is the original instance and terminal states are those for which $T{=}\emptyset$.

Actions. An action is an assignment (I, t) of a non-empty set of agents $I{\subset}A$ to a single task $t{\in}T$. Thus, the solution is built by iteratively assigning the tasks, one by one until exhaustion.

Transitions. It is intuitive to see that once a task has been assigned, the remaining optimization subproblem is of the same nature as the original one, with one less task to assign and some adjustments of the problem parameters. More precisely, the transition after taking action (I, t) can be written as:

$$\text{MATAP}\langle A, T, \delta, \lambda\rangle \xrightarrow{(I,t)} \text{MATAP}\langle A, T', \delta', \lambda'\rangle, \quad (2)$$

where the selected task t is discarded: $T'{=}T\backslash\{t\}$. Most of the instance parameters remain unchanged, except the parameters that are linked to the initial tasks of the selected agents. In fact, the initial tasks are updated to account for task t in the following way. First, the processing time of the initial tasks for the assigned agents become:

$$\delta'_{i(a)} = \max_{a'\in I}(\delta_{i(a')}{+}\delta_{a't}){+}\delta_t \quad \forall a{\in}I, \quad (3)$$

to account for all the agents in I finishing their initial tasks ($\delta_{i(a')}$), then moving to task t ($\delta_{a't}$) and finally jointly executing it (δ_t). Second, the transfer times for these agents to start a new task v after completing their initial task become:

$$\delta'_{av}{=}\delta_{atv} \quad \forall a{\in}I, v{\in}T'. \quad (4)$$

And finally the initial dependency parameters become:

$$\lambda'_{av}{=}\lambda_{stv} \quad \forall a{\in}A_s{\cap}I, v{\in}T'. \quad (5)$$

Note that this transition to an instance of the MATAP proves that our problem does indeed satisfy assumption (i) of the BQ-NCO framework.

Valid Actions. To be valid, an assignment action (I, t) should on one hand satisfy the constraints of the MATAP and on the other hand be such that the remaining sub-problem (i.e. the next state) is feasible. We formalize this using the following conditions:

$$\begin{array}{ll|ll} \forall s{\in}S & \lambda_{st}{=}|A_s{\cap}I|, & \forall a{\in}I, v{\in}T' & \lambda_{av}{=}0, \\ \forall a{\in}A\backslash I & \lambda_{at}{=}0, & \forall s{\in}S, u{\in}T' & \lambda_{sut}{=}0. \end{array} \quad (6)$$

The first condition ensures that the skill requirements of task t are satisfied by the agents I; the second and third ensure that there is no agent outside of I which was required to be been assigned to t and that the agents in I were not required by other tasks than t; and finally the last condition ensures that there is no remaining task u that should be assigned before t. These conditions are obviously necessary to ensure the current constraints and the feasibility of the remaining subproblem. One can prove that they are also sufficient (proof not included for lack of space).

Reward. The reward is based on the incremental change of the objective caused by the assignment, such that at the end of the construction process, the (undiscounted) sum of rewards corresponds exactly to the value of the objective for the whole solution. When the objective is the sum of completion times, we define the reward as minus the completion time z_t of the newly assigned task t, i.e. $r{=}{-}z_t{=}{-}\max_{a\in I}(\delta_{i(a)}{+}\delta_{at}){+}\delta_t$. When the objective is the makespan, the incremental change in the makespan can be either 0 if the completion time of t is smaller than the completion time of one of the initial tasks, or the difference otherwise. Therefore we can define the reward as $r{=}{-}\max(z_t{-}\max_{a\in A}\delta_{i(a)}, 0)$.

3.3. Uncertainty, dynamic changes, and re-planning

The MATAP instances we have considered so far are static (fixed number of tasks) and assume full knowledge of the time parameters of the problem, namely the task durations δ_u and transfer times δ_{auv}. In practice though, these quantities are always uncertain and can only be estimated. To address this uncertainty as well as the dynamic nature of the problem (tasks may arise at any time), we propose a simple re-planning scheme. Given an initial schedule, computed based on the estimated durations, the agents start applying the plan. Every time a task is completed (by its assigned set of agents), we compare its actual completion time with the planned completion time. If the difference is above a certain threshold, or if new tasks have appeared, we trigger the *re-planning*. To do so, we use the most up-to-date information to define a new MATAP instance as follows. The already-executed tasks are excluded and the ongoing tasks are accounted for by updating the durations and dependencies of the agents' initial tasks, similarly to the MDP transition above. By solving the new instance, we get an updated plan that provides the next assignments. We continue checking for new tasks or completion time deviations and re-plan accordingly each time a task is finished.

4. Model Architecture and Training

In order to solve the MATAP MDP, we propose to learn a neural-network parametrized policy. In this section, we first describe the state representation that we adopt as input to the policy model, then the model architecture and finally the training procedure.

4.1. State Representation

We represent the MDP state, i.e. a MATAP instance $\langle A, \bar{T}, \delta, \lambda \rangle$ as a graph which involves two types of nodes, agents and tasks, as well as two types of edges, task-task and agent-task (there is no agent-agent edges). The features for each node and edge types are defined as follows.

Node/Edge	Features
Agent a	$(\mathbb{1}_{\{a \in A_s\}})_{s \in S}$: one-hot skill indicator vector $\delta_{i(a)}$: completion time of its initial task
Task u	δ_u: processing time $(\lambda_{su})_{s \in S}$: required skills
Agent-task a-u	δ_{au}: initial transfer time λ_{au}: initial dependency constraint indicator
Task-task u-v	$(\delta_{puv})_{s \in S}$: transfer time $(\lambda_{suv})_{s \in S}$: precedence constraint indicator

Note that in our application agent transfer times between tasks δ_{auv} are homogeneous within each category of agents $s \in S$ (hence the notation δ_{suv} in the table). Therefore all the dimensions of the features above are independent from the number of agents and tasks, which will allow us to learn a unique policy that can be applied to variable number of tasks and agents. The number of skills however is fixed.

4.2. Model Architecture

To solve our problem, we adopt the Generalist combinatorial optimization model GOAL [7], which provides state-of-the-art performance for a variety of classical CO problems and has been shown to generalize nicely beyond its training distribution. This is an important aspect for us since in practice the number of available agents and tasks varies everyday. GOAL is a transformer-based architecture with a sequence of layers featuring multi-head mixed-attention blocks, adapted to handle data represented as dense multi-partite graphs with both node and edge features. While we use the original *backbone*, we define a custom *encoder* and *decoder* tailored to MATAP. Figure 2 illustrates the architecture of the model. Given a MATAP instance as input, the encoder projects the instance features into tasks, agents and relationships embeddings. The transformer backbone processes these embeddings and outputs a final embedding for each task and agent. These are then processed by the decoder to return a distribution over the tasks in T and the different families of agents $(A_s)_{s \in S}$. In order to make the agent's scores conditioned on the selected task, we introduce the following mechanism. The decoder first computes the tasks scores, then selects a task (either by sampling from the scores or taking the arg max), then the embedding of the selected task is concatenated to the embeddings of each agent, and finally the agents scores are computed based on this richer representation. We use appropriate masks on the tasks and agents to restrict the assignments to the *valid actions* defined in Section 3.2.

4.3. Self-Supervised Training

The neural model GOAL has been trained by imitation of high-quality solutions, provided by specialized solvers, for a number of classical CO problems. However, for our real-life MATAP there are no specialized solvers and even generic mixed-integer programming solvers would be out of reach for realistic instance sizes. A natural alternative would be to use Reinforcement Learning (RL), but previous studies have shown the challenges of applying RL to this kind of "heavy" transformer architectures [20]. Recently, a self-supervised training paradigm has been successful in neural combinatorial optimization [6,25]. Inspired by those works, we propose a training that alternates between imitation and improvement steps. Starting from a randomly initialized policy, we sample a number of solutions for each training instance. The best solution for each instance is then passed to the model for imitation. As we update the model, the next sampled solutions should improve, creating a virtuous cycle of self-improvement. In our experiments, we observed that although successful this approach requires extensive sampling at each step, making it computationally expensive. In order to mitigate this training cost, we start by pre-training our model by imitation of simple heuristic solutions, then we fine-tune it using the self-supervised approach.

5. Experiments

In this section, we describe our simulation environment and experimental setup before a quantitative evaluation of our approach in both a deterministic and stochastic setting.

5.1. Data Center Simulation

Agents and Tasks. In addition to warehouse robots (WRs), delivery robots (DRs) and human workers (HWs), we consider workstations (WS) as "agents" since they are a limited resource required to transfer assets between WRs and DRs. We consider three types of collaborative tasks:

(i) Loading tasks: a WR retrieves a set of assets from designated shelves and loads them onto a DR at a WS,

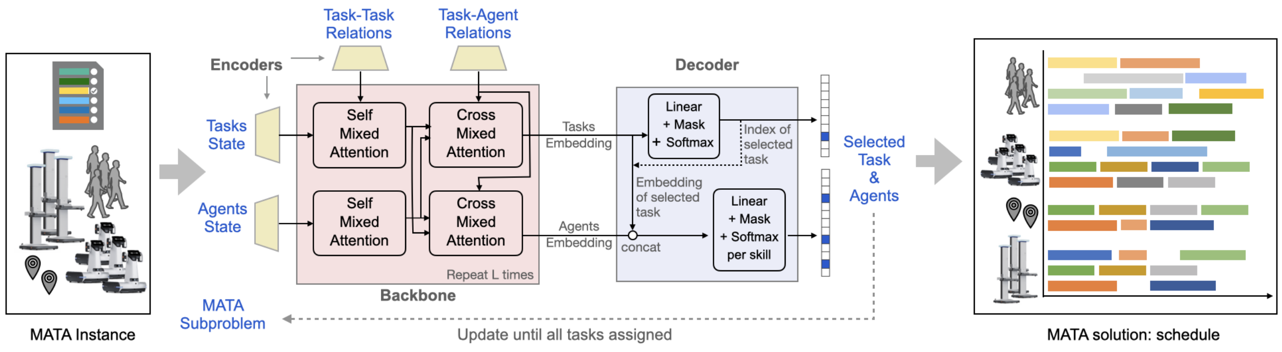

Figure 2: Overview of our end-to-end learning approach for the Multi-Agent Task Assignment problem. A MATA instance defines the input task, agent and relations state which go through a linear encoder to get their initial embeddings. These are processed by a multi-layer transformer-based backbone to get final tasks and agents embeddings. The latter are used in a decoder to output a task and a set of agents conditioned on the task. This assignment is used to define a MATA subproblem and iterate until all the tasks have been assigned. We can then construct a schedule.

making multiple trips between the shelves and the WS;

(ii) Unloading tasks: a WR picks up a set of assets from a DR at a WS and places at designated shelves;

(iii) Processing tasks: a HW meets a DR at a designated server rack to process some assets (e.g. installing a new server delivered by the DR and/or loading a faulty server onto it).

A *ticket* corresponds to a sequence of such tasks which introduces the precedence constraints. For example, a replacement ticket is decomposed into a loading task u, a processing task v and an unloading task u' which must be executed in this order while sharing the same DR.

Environment Simulation. We have developed a simulator that accurately models key aspects of Naver's GAK Sejong data center, focusing on the factors that influence maintenance and installation ticket workflows. Our simulation includes a warehouse and twelve server rooms distributed across multiple floors. To ensure realism, we use the precise facility map, as well as the physical characteristics and motion profiles of Naver's real robots, SeRo and GaRo, to simulate their movements accurately.

To key inputs to define a MATAP instance are the task durations and transfer times, which include the robots and workers travel times, the loading and unloading times as well as the assets processing times. Specifically, for DRs, trip durations between the warehouse workstations and various locations in the server rooms are computed using real paths, robot motion profiles, and estimated elevator ride times based on actual measurements. WR movements within the warehouse, including retrieving assets from shelves and delivering them to workstations, are simulated using a custom multi-agent path-finding algorithm [18] that accounts for precise WR dynamics and generates collision-free trajectories in the constrained warehouse environment. Finally, loading and unloading actions are simulated in detail to accurately estimate their durations, while worker processing times (e.g., server installation and replacement) are derived from historical data. By incorporating these real-world data and measurements, our simulation closely reflects the operational conditions of a real data center.

5.2. Experimental Setup

Baselines. We adapt two standard scheduling baselines from the family of Priority Dispatching Rules heuristics [11], which are widely used in real-world scheduling applications. These heuristics greedily assign jobs to the best machine based on a predefined priority order. Two common priority rules are the *Shortest Processing Time* (SPT) and *Longest Processing Time* (LPT), which are particularly suited for minimizing the makespan $C_{\max}$ and the sum of completion times C_{sum}, respectively. We also consider the naive rule of *First Come First Serve* (FCFS) as a reference. To adapt these heuristics to MATAP, we group tasks per tickets and aggregate their parameters, then process the tickets sequentially according to the chosen priority rule. For each sub-task of the ticket (in the order of the precedence constraint, if any), we assign the agent within each required category that can begin execution the earliest, keeping the same agent for the sub-tasks that are linked by a dependency constraint.

Model Hyperparameters and Hardware Our model

Table 1: Average reduction of the $C_{\max}$ and C_{sum} on MATAP test instances from the training distribution (underlined) and with various numbers of tasks and agents.

	Tickets	Agents				FCFS		LPT		OURS		
										$\pi^{\text{ss}}_{C_{\max}}$	$\pi^{\text{pf}}_{C_{\max}}$	
		DR	HW	WR	WS	reduction	time	reduction	time	reduction		time
	40	8	8	3	2	0.00%	15s	-4.98%	17s	-11.19%	**-11.93%**	50s
	20					0.00%	7s	-7.12%	8s	-12.25%	**-12.48%**	18s
	60					0.00%	24s	-4.74%	27s	-9.65%	**-10.08%**	1.2m
	80	8	8	3	2	0.00%	35s	-4.51%	40s	-8.21%	**-8.30%**	2.5m
$C_{\max}$	100					0.00%	46s	-4.03%	51s	-6.95%	**-7.42%**	3.8m
	200					0.00%	2.1m	-3.72%	2.3m	-5.15%	**-5.26%**	14.5m
		10	10	3	2	0.00%	15s	-5.34%	15s	-16.05%	**-16.91%**	50s
	40	6	6	2	2	0.00%	15s	-4.54%	15s	-6.83%	**-7.59%**	50s
		4	4	2	1	0.00%	15s	-4.65%	15s	-6.13%	**-7.01%**	50s
								SPT		$\pi^{\text{ss}}_{C_{\text{sum}}}$	$\pi^{\text{pf}}_{C_{\text{sum}}}$	
	40	8	8	3	2	0.00%	15s	-7.87%	17s	-20.49%	**-21.19%**	50s
	20					0.00%	7s	-5.49%	8s	-17.78%	**-18.09%**	18s
	60					0.00%	24s	-8.09%	27s	-17.94%	**-20.50%**	1.2m
	80	8	8	3	2	0.00%	35s	-9.07%	40s	-18.89%	**-20.85%**	2.8m
C_{sum}	100					0.00%	46s	-8.74%	51s	-18.61%	**-20.27%**	3.8m
	200					0.00%	2.1m	-9.08%	2.3m	-13.26%	**-19.57%**	14.5m
		10	10	3	2	0.00%	15s	-7.34%	15s	**-24.03%**	-23.93%	50s
	40	6	6	2	2	0.00%	15s	-6.65%	15s	-16.55%	**-18.94%**	50s
		4	4	2	1	0.00%	15s	-6.86%	15s	-16.03%	**-19.44%**	50s

consists of 9 layers, each with mixed-attention blocks of 8 heads, an embedding size of 128, a feed-forward dimension of 512, and ReZero normalization [1]. All experiments are conducted on a single NVIDIA V100-SXM2 GPU (32GB RAM).

Training Data. We train our model with realistic instances, based on distributions defined using historical data and our data center simulator. We consider instances with 8 DRs, 8 HWs, 3 WRs, 2 workstations and 40 tickets, which result in approximately 90–110 tasks per instance. For the supervised pre-training, we generate solutions to 30,000 instances using SPT or LPT depending on the intended objective.

Training Details. We train four policies: $\pi^{\text{ss}}_{C_{\max}}$, $\pi^{\text{pf}}_{C_{\max}}$, $\pi^{\text{ss}}_{C_{\text{sum}}}$ and $\pi^{\text{pf}}_{C_{\text{sum}}}$, optimizing for either the $C_{\max}$ or C_{sum} objectives, using self-supervised training or a combination of supervised pretraining followed by self-supervised fine-tuning (as described in Sec 4.3). In the case of supervised pre-training, we train the models for 10 epochs with batches of 128 instances. For the self-supervised training and fine-tuning, at each training step, we randomly generate 4 MATAP instances from our realistic distributions, sample 64 solutions from the current policy, and use the best one for the next iteration of imitation learning. All models are trained to convergence using the Adam optimizer [13] with a learning rate of 0.0001 and a decay rate of 0.99 every 10 steps. Self-supervised training takes 132 hours, while supervised pretraining completes in 1.8 hours, with an additional 49 hours for self-supervised fine-tuning.

5.3. Evaluation in a Deterministic Setting

We first evaluate our learned policy in a deterministic setting, where we assume that all the problem parameters are known in advance. Fig. 3 shows the GANNT charts obtained by our model (top) vs a heuristic (bottom) on a randomly generated problem instance, illustrating the performance gain on that instance (makespan reduced by 8.9%). Quantitatively, Table 1 compares our model's average performance against the FCFS baseline, which serves as a reference, as well as the SPT and LPT heuristics. In addition to evaluating the models on test instances drawn from the training distributions (underlined parameters), we assess their generalization performance by varying the number of tickets and agents. Each test dataset contains 128 instances, and we report the average performance and total computation time. For both objectives and all scenarios, we observe that our model consistently outperforms the standard scheduling heuristics. As expected, performance decreases as the test distributions deviate further from the training distribution (e.g. 200 versus 40 tickets, i.e. approximately 500 versus 100 tasks), but it remains superior to the baselines. Finally we observe that the computation time scales with the number of tasks. This is expected as the policy is called as many times as there are tasks (Fig. 2). In practice, the agents can start executing the plan after a few assignments, while the remaining ones are still being computed.

5.4. Evaluation in a Stochastic Setting

We now evaluate our policies in a more realistic setting, where only estimates of the task durations $(\delta_u)_{u\in T}$ and transfer times $(\delta_{auv})_{a\in A,u,v\in T}$ are available for planning. The true realizations of these parameters are revealed once the tasks are executed and are used to evaluate the quality of the plans (here the sum of completion times). We consider three scenarios:

(i) Looking-into-the-future: The policy is applied with the true realizations as input. This provides a lower bound on the objective, which will serve as a reference for computing a regret metric.

(ii) Planning Once: The policy is applied using estimates as inputs, and its performance is evaluated with the true realizations.

(iii) Regular Re-planning: The policy is initially applied using estimates as inputs. As we start executing the plan, if a significant deviation (above a threshold) is detected between the planned completion time of a task and its actual completion time, the state is updated using the true realizations up to that point. The policy is then re-applied to the updated (more accurate) state, with the estimated durations for the remaining tasks.

We define the regret measure as the relative gap between a given policy's total completion time and that of the looking-into-the-future policy. It represents how much worse the policy performs compared to an ideal policy which has full knowledge of the environment in hindsight. Table 2 compares the regret of the planning-once and re-planning policies, on two data distributions for the uncertain parameters, one with low variance and one with high variance. For each scenario, we generate 128 instances for different numbers of tickets and report the average regret and number of re-plannings (rounded to the nearest integer), for different re-planning thresholds.

As expected, a higher variance leads to a significant increase in the regret for the planning-once policy while re-planning helps mitigate this effect: the lower the threshold, the better the regret. Interestingly, replanning with a threshold of 10 or 7.5 minutes is sufficient to achieve a very reasonable regret, with consistent performance across both objectives and different numbers of tickets. Regarding the re-planning time, as mentioned above, agents can start executing the plan as soon as their first assignment is computed, which takes at most a few seconds. These results confirm the effectiveness of the re-planning approach in handling real-world uncertainties.

6. Conclusion

In this paper, we presented an end-to-end learning framework for solving a complex real-world multi-agent task assignment problem, involving multi-robot tasks and cross-schedule dependencies. Despite the complex dependency constraints, we proposed a formulation of the underlying optimization problem that respects the optimality principle of dynamic programming, enabling the definition of an efficient Markov Decision Process to model the construction of optimal solutions. Using a compact graph representation of the problem, without any feature engineering, we demonstrated that a transformer-based policy can be effectively learned, through a combination of supervised pre-training and self-supervised fine-tuning. Using a realistic data center simulator, we showed that our learned policies outperform standard scheduling baselines while generalizing to various numbers of tasks and agents. Finally we proposed a simple yet effective re-planning strategy to deal with the inherent uncertainties of real-world systems. Beyond the addressed data center use-case, we believe that our approach provides a generic framework to learn customized policies that can be applied to a variety of multi-agent task assignment problems. As robotics services continue to expand, such data-driven approaches to multi-agent task assignment and scheduling will be crucial for the overall efficiency of such services.

Table 2: Performance (average regret, the lower the better) of the learned policy with re-planning.

		Low Variance Scenario					High Variance Scenario				
	Tickets	Planning once	Re-planning 20m	15m	10m	7.5m	Planning once	Re-planning 20m	15m	10m	7.5m
$C_{\max}$	40	6.99% (0)	4.71% (1)	3.22% (3)	2.09% (7)	**1.28**% (13)	14.72% (0)	5.73% (5)	3.29% (11)	1.95% (20)	**1.68**% (27)
	60	7.13% (0)	4.65% (2)	3.46% (4)	1.26% (11)	**0.73**% (19)	15.80% (0)	5.48% (8)	3.38% (16)	1.57% (30)	**1.24**% (42)
	80	7.11% (0)	4.82% (3)	3.36% (6)	1.38% (15)	**1.01**% (25)	15.57% (0)	4.94% (12)	2.40% (22)	1.83% (40)	**0.95**% (55)
	100	7.05% (0)	4.19% (4)	2.81% (7)	1.38% (19)	**0.76**% (31)	15.22% (0)	4.88% (15)	2.35% (26)	1.03% (49)	**0.35**% (68)
C_{sum}	40	5.16% (0)	3.78% (1)	3.16% (3)	1.98% (7)	**1.16**% (12)	12.64% (0)	6.46% (6)	4.38% (10)	2.22% (19)	**1.66**% (28)
	60	5.95% (0)	3.96% (2)	3.55% (4)	2.04% (11)	**1.17**% (18)	14.41% (0)	7.07% (8)	5.25% (16)	3.00% (30)	**2.56**% (41)
	80	6.34% (0)	4.55% (3)	3.52% (5)	1.97% (15)	**1.29**% (25)	15.34% (0)	7.28% (12)	4.77% (21)	2.95% (38)	**2.42**% (55)
	100	6.68% (0)	4.63% (4)	3.89% (7)	2.24% (19)	**1.32**% (31)	16.79% (0)	7.82% (15)	4.45% (33)	3.17% (48)	**3.02**% (69)

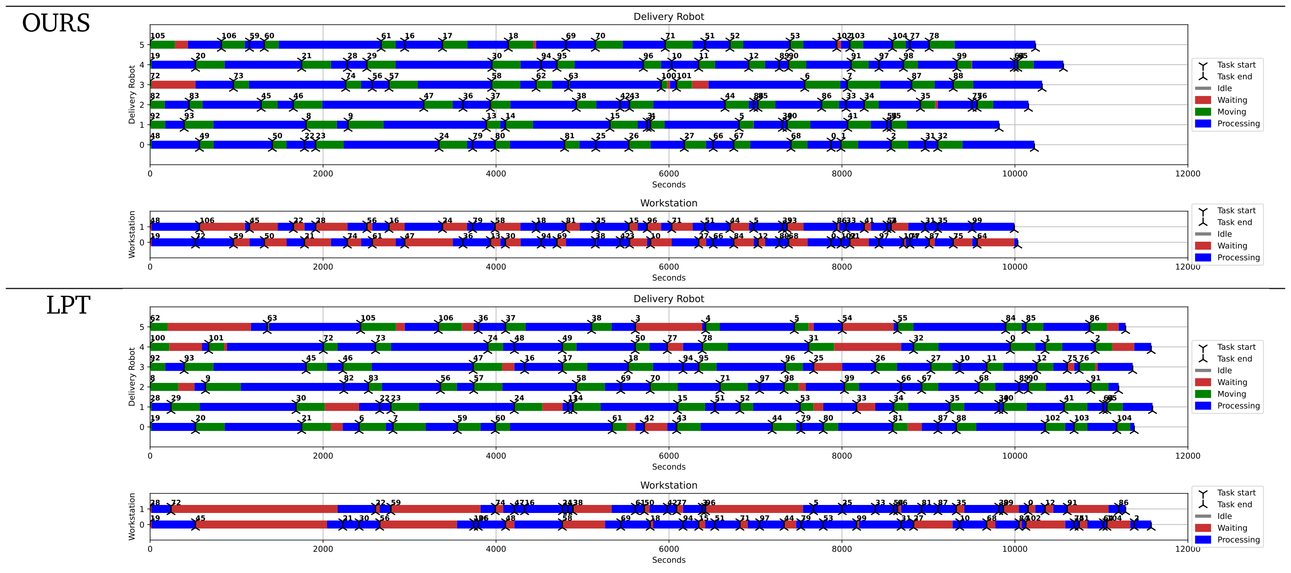

Figure 3: GANNT charts for our model ($\pi^{\text{pf}}_{C_{\max}}$) vs LPT for minimum makespan on one instance. Only 2 agent types are represented.

References

[1] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. ReZero is all you need: Fast convergence at large depth. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 1352–1361. PMLR, December 2021. 7

[2] Richard Bellman. The Theory of Dynamic Programming. Technical Report P-550, The Rand Corporation, Santa Monica, CA, U.S.A., July 1954. 4

[3] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. 2

[4] Dimitri Bertsekas. *Dynamic Programming and Optimal Control: Volume I*, volume 1. Athena Scientific, 2012. 4

[5] Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial Optimization and Reasoning with Graph Neural Networks. In *Twenty-Ninth International Joint Conference on Artificial Intelligence*, volume 5, pages 4348–4355, August 2021. 2

[6] Andrea Corsini, Angelo Porrello, Simone Calderara, and Mauro Dell'Amico. Self-Labeling the Job Shop Scheduling Problem. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 105528–105551. Curran Associates, Inc., 2024. 5

[7] Darko Drakulic, Sofia Michel, and Jean-Marc Andreoli. GOAL: A Generalist Combinatorial Optimization Agent Learner. In *International Conference on Learning Representations*, Singapore, 2025. 3, 5

[8] Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. BQ-NCO: Bisimulation Quotienting for Efficient Neural Combinatorial Optimization. In *Advances in Neural Information Processing Systems*, volume 36, pages 77416–77429, December 2023. 3

[9] Brian P. Gerkey and Maja J. Matarić. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *The International Journal of Robotics Research*, 23(9):939–954, September 2004. 2

[10] Hamza Chakraa, François Guérin, Edouard Leclercq, and Dimitri Lefebvre. Optimization techniques for Multi-Robot Task Allocation problems: Review on the state-of-the-art. *Robotics and Autonomous Systems*, 168:104492, October 2023. 2

[11] R. Haupt. A survey of priority rule-based scheduling. *Operations-Research-Spektrum*, 11(1):3–16, March 1989. 6

[12] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. Multi-robot Task Allocation: A Review of the State-of-the-Art. In Anis Koubâa and J.Ramiro Martínez-de Dios, editors, *Cooperative Robots and Sensor Networks 2015*, pages 31–51. Springer International Publishing, Cham, 2015. 2

[13] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, San Diego, May 2015. 7

[14] Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems! In *International Conference on Learning Representations*, September 2018. 2

[15] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, October 2013. 2

[16] Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 34, pages 5138–5149. Curran Associates, Inc., 2021. 2, 3

[17] Christopher Leet, Chanwook Oh, Michele Lora, Sven Koenig, and Pierluigi Nuzzo. Task Assignment, Scheduling, and Motion Planning for Automated Warehouses for Million Product Workloads. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7362–7369, October 2023. 2

[18] Vassilissa Lehoux-Lebacque, Tomi Silander, Christelle Loiodice, Seungjoon Lee, Albert Wang, and Sofia Michel. Multi-Agent Path Finding with Real Robot Dynamics and Interdependent Tasks for Automated Warehouses. In *European Conference on Artificial Intelligence*, 2024. 6

[19] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, pages 1898–1900, Richland, SC, May 2020. Interna-

tional Foundation for Autonomous Agents and Multiagent Systems. 2

[20] Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural Combinatorial Optimization with Heavy Decoder: Toward Large Scale Generalization. In *Advances in Neural Information Processing Systems*, volume 36, pages 8845–8864, 2023. 5

[21] Hang Ma, Jiaoyang Li, T.K. Satish Kumar, and Sven Koenig. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 837–845, Richland, SC, May 2017. International Foundation for Autonomous Agents and Multiagent Systems. 2

[22] James Orr and Ayan Dutta. Multi-Agent Deep Reinforcement Learning for Multi-Robot Applications: A Survey. *Sensors*, 23(7):3625, January 2023. 2

[23] Bumjin Park, Cheongwoong Kang, and Jaesik Choi. Cooperative Multi-Robot Task Allocation with Reinforcement Learning. *Applied Sciences*, 12(1):272, January 2022. 2

[24] Steve Paul, Payam Ghassemi, and Souma Chowdhury. Learning Scalable Policies over Graphs for Multi-Robot Task Allocation using Capsule Attention Networks. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8815–8822, May 2022. 2

[25] Jonathan Pirnay and Dominik G. Grimm. Self-improvement for neural combinatorial optimization: Sample without replacement, but improvement. *Transactions on Machine Learning Research*, 2024. 3, 5

[26] Yara Rizk, Mariette Awad, and Edward W. Tunstel. Cooperative Heterogeneous Multi-Robot Systems: A Survey. *ACM Comput. Surv.*, 52(2):29:1–29:31, April 2019. 2

[27] Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning Search Space Partition for Black-box Optimization using Monte Carlo Tree Search. *Advances in Neural Information Processing Systems*, 33:19511–19522, 2020. 2

[28] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Multi-Decoder Attention Model with Embedding Glimpse for Solving Vehicle Routing Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13):12042–12049, May 2021. 2

[29] Qinghong Xu, Jiaoyang Li, Sven Koenig, and Hang Ma. Multi-Goal Multi-Agent Pickup and Delivery. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9964–9971, October 2022. 2

[30] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, pages 1621–1632, Red Hook, NY, USA, December 2020. Curran Associates Inc. 2, 3