# Adaptive Obstacle-Aware Task Assignment and Planning for Heterogeneous Robot Teaming

Nan Li, Jiming Ren, Haris Miller, Samuel Coogan, Karen M. Feigh, and Ye Zhao

***Abstract*—Multi-Agent Task Assignment and Planning (MATP) has attracted growing attention but remains challenging in terms of scalability, spatial reasoning, and adaptability in obstacle-rich environments. To address these challenges, we propose OATH — Adaptive Obstacle-Aware Task Assignment and Planning for Heterogeneous Robot Teaming — which advances MATP by introducing a novel obstacle-aware strategy for task assignment. First, we develop an adaptive Halton sequence map, the first known application of Halton sampling with obstacle-aware adaptation in MATP, which adjusts sampling density based on obstacle distribution. Second, we propose a cluster–auction–selection framework that integrates obstacle-aware clustering with weighted auctions and intra-cluster task selection. These mechanisms jointly enable effective coordination among heterogeneous robots while maintaining scalability and suboptimal allocation performance. In addition, our framework leverages an LLM to interpret human instructions and directly guide the planner in real time. We validate OATH in both NVIDIA Isaac Sim and real-world hardware experiments using TurtleBot platforms, demonstrating substantial improvements in task assignment quality, scalability, adaptability to dynamic changes, and overall execution performance compared to state-of-the-art MATP baselines. A project website is available at https://llm-oath.github.io/.**

***Note to Practitioners*—Coordinating heterogeneous robot team in dynamic, obstacle-rich environments remains a major challenge in practical robotics. This paper presents OATH, a hierarchical framework that integrates adaptive and obstacle-aware task assignment and planning for heterogeneous robot teams. The practical contributions are threefold. First, we propose an adaptive Halton sequence map that automatically adjusts sampling density based on obstacle distribution. Combined with Dijkstra-based distances, this map provides realistic task-to-task costs, enabling robots to achieve high-quality task assignments while maintaining scalability. Second, we develop a heterogeneous cluster–auction–selection framework that reduces allocation complexity while respecting robot capacity and capability constraints, which are common in real industrial deployments. Third, we implement an LLM-guided interaction module that interprets natural language commands and supports real-time replanning during task execution, allowing human operators to dynamically adjust system behavior when needed. The proposed framework is validated in both NVIDIA Isaac Sim and real-world hardware experiments using TurtleBot platforms. The experimental results show consistent allocation behavior, stable replanning, and reliable execution under realistic communication latency and sensing noise. These findings suggest that the proposed framework remains effective beyond simulation settings and can be implemented on physical multi-robot systems.**

The authors are with the Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA. Email: nan.li@gatech.edu

This work is sponsored by Lockheed Martin Corporation University Research program. The work is that of the authors and does not represent an official position of LMCO.

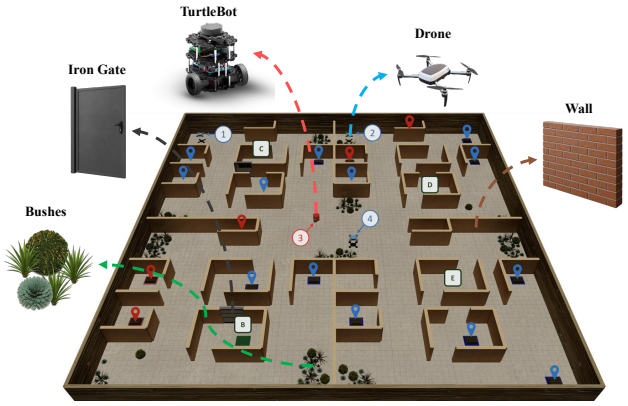



Fig. 1. Problem setup in the Isaac simulation environment. A heterogeneous robot team of ground robots and drones is deployed in a maze-like environment. Two types of tasks are represented by blue and red markers, while delivery rooms are labeled **B–E**. The environment contains both known obstacles (walls) and initially unknown obstacles (iron gates and bushes).

***Index Terms*—Multi-robot systems, Heterogenous robot teaming, Task assignment, Large Language Models (LLMs)**

## I. Introduction

Multi-robot systems (MRSs) have attracted growing attention for their potential to enhance scalability, adaptability, and fault tolerance, yet they still present significant challenges in coordination, task assignment, and planning. Effective cooperation among multiple robots is essential for accomplishing complex objectives in domains such as agriculture, surveillance, search and rescue, and warehouse operations [1]–[5]. A central challenge among many MRSs is the integration of task assignment and path planning to ensure coordinated and efficient operation [1], [6]–[8].

In multi-robot scenarios (e.g., Fig 1), global missions are often decomposed into sub-tasks and distributed among individual agents. A widely studied formulation is the Multi-Agent Pickup and Delivery (MAPD) problem [9], which is common in real-world settings such as warehouse automation and logistics. Efficiently solving MAPD is essential for achieving scalability and coordination. MAPD generally involves two main components: task assignment and path planning.

For task assignment, two widely used approaches are auction-based methods [10]–[12] and optimization-based methods [1]. Some works combine the two, for example by first clustering tasks with optimization techniques and then using auctions to allocate task clusters to robots [13], [14]. This hybrid method has proven effective for large-scale problems

and improves both scalability and allocation quality. Our work follows this paradigm but introduces two key improvements: first, obstacle information is directly incorporated into the task assignment process; second, a hierarchical allocation framework is designed to enhance heterogeneity handling, efficiency, and scalability.

To achieve obstacle-aware task assignment, we develop an adaptive Halton sequence map. The Halton sequence is a classical sampling method that has been applied in robot motion and path planning to achieve more uniform coverage and smoother trajectories compared to grid-based maps [15], [16]. Prior works typically employ Halton or other quasi-random sequences for uniform roadmap construction or to improve sampling coverage in Probabilistic Roadmap Methods (PRM) and Rapidly-exploring Random Tree (RRT) style planners [17]. However, these methods do not adapt the point distribution to the presence of obstacles. Recent studies on obstacle-aware sampling often rely on hard rejection (discarding points within obstacle regions) or potential-field-based biasing [18]. However, these approaches lack a principled way to adapt sampling density according to obstacle distribution.

To enable this principled mechanism, we extend Halton sampling into an adaptive obstacle-aware version, which to our knowledge is the first application of Halton with obstacle-aware adaptation in the Multi-Agent Task Assignment and Planning (MATP) setting. Specifically, we introduce a probabilistic acceptance function that adjusts point density according to obstacle proximity, which yields sparse sampling in open areas and dense sampling in cluttered regions. We then apply Dijkstra's algorithm on the adaptive Halton sequence map to compute obstacle-feasible paths, which are subsequently used in the task assignment stage to explicitly account for obstacle constraints.

To improve efficiency and scalability in MATP, particularly for heterogeneous robot teams, we propose a hierarchical task assignment framework based on a cluster–auction–selection structure. This framework leverages the scalability of the cluster–auction approach while mitigating its suboptimality by incorporating obstacle-aware distances during clustering [14], [19]. After cluster assignment, an additional intra-cluster optimization step is undertaken to select specific tasks, combining the strengths of auction-based and optimization-based methods. By restricting the optimization to a smaller intra-cluster scale, the framework also alleviates the computational overhead typically associated with global, large-scale optimization [1]. Overall, this hierarchical design achieves a balance between optimality, scalability, and efficiency for heterogeneous robot teams.

While adaptive sampling addresses obstacle-aware task assignment and the hierarchical task allocation framework enhances efficiency and scalability, another critical aspect of practical multi-robot systems is the ability to incorporate real-time human instructions or intervention.

Recent advances in Large Language Models (LLMs), such as GPT-4 [20], LLaMa 3 [21], and Gemini 1.5 [22], have demonstrated strong capabilities in natural language reasoning and generalization [23]. These abilities open new possibilities for human–robot interaction in multi-robot systems. Existing research has leveraged LLMs either as translators, converting natural language instructions into formal task representations [24], [25], or as high-level planners for generating mission-level task structures [26], [27]. However, such approaches are typically limited to pre-mission planning and do not support online interaction during task execution.

In real deployments, human operators often need to introduce new tasks, adjust priorities, or impose additional safety constraints during execution. Existing research on human–robot interaction in multi-robot system primarily falls into three categories [28]: humans provide instructions only at the start of the mission, after which the robots execute tasks autonomously [29]; humans give commands only after a task or robot failure occurs, when intervention is needed for recovery [30]; or interactive platforms allow humans to query robot status or ask clarifying questions, but not to modify behavior in real time [31]. To the best of our knowledge, there is still no MATP framework that allows humans to inject new instructions during execution and enables robots to immediately replan in response.

To address all above limitations, we propose **OATH**: Adaptive **O**bstacle-**A**ware **T**ask Assignment and Planning for **H**eterogeneous Robot Teaming, which achieves adaptivity in three aspects: obstacle-aware sampling, capability-aware task assignment, and real-time replanning based on human instructions. Our main contributions are as follows:

- We propose an **adaptive Halton sequence map**, where sampling density automatically adjusts according to obstacle distribution.
- We develop a **hierarchical heterogeneous allocation framework** based on a cluster–auction–task selection structure, which generalizes to any number of task types and reduces allocation complexity while maintaining capacity and capability constraints.
- We present a **fully integrated online pipeline** with LLM-guided interaction for multi-robot system. It interprets natural language inputs and supports real-time replanning in response to human instructions.
- We validate the OATH framework through extensive simulation case studies and real-world hardware experiments, showing substantial improvements in task assignment quality, scalability, adaptability to dynamic changes, and overall execution performance compared to state-of-the-art MATP baselines.

## II. Related Work

### A. Task Assignment

Multi-robot Systems (MRSs) are prevalent in applications such as search and rescue, pickup and delivery, and target detection. Multi-robot task assignment focuses on assigning tasks to robot teams to optimize metrics like mission completion time. These assignment methods are typically categorized into centralized and decentralized approaches.

**Centralized algorithms** provide near-optimal solutions but often rely on stable communication and global information. Early work explored metaheuristics such as genetic algorithms

for UAV task assignment with precedence and timing constraints [32], while later studies focused on improving scalability through approaches like multi-objective particle swarm optimization for coalition formation in disaster response [33] and federated optimization for dynamic ridesharing [34]. Other efforts have developed heuristic methods for cooperative scheduling under precedence constraints [35].

Despite these advances, centralized methods require reliable communication. To address this, Otte et al. [36] proposed auction-based techniques that maintain performance under limited connectivity, serving as a middle ground between centralized and distributed planning.

**Decentralized approaches** rely on local information, offering better scalability and robustness in dynamic or communication-limited settings, though often at the cost of optimality. Auction-based methods are widely studied, including consensus-based bidding frameworks [10], extensions with conflict resolution mechanisms [37], and group-based auctions that handle capacity and time constraints [38]. Other strategies focus on task updates in dynamic environments, such as greedy reallocation based on marginal cost changes [39].

**Learning-based approaches** have been increasingly adopted to enhance adaptability and scalability in multi-robot task assignment in recent research. Deep reinforcement learning (DRL) and graph neural network (GNN)-based methods enable agents to learn allocation strategies directly from experience, allowing generalization beyond handcrafted heuristics. Representative works include attention-inspired DRL frameworks for large-scale warehouse allocation [40], graph-based multi-agent reinforcement learning for heterogeneous teams [41], and neural combinatorial optimization approaches using attention or GNN architectures [42].

While these learning-based methods demonstrate impressive adaptability, they often struggle to guarantee conflict-free task assignment or consistent convergence, occasionally yielding lower success rates compared to model-based or auction-based methods. For example, learning-enhanced consensus schemes such as GCN-augmented Consensus-Based Bundle Algorithm (CBBA) [43] combine neural scoring with classical distributed auctions, but still rely on heuristic conflict resolution.

In our framework, we combine centralized and decentralized task assignment. A centralized cluster planner first performs task clustering based on global information. Each robot is then assigned to a specific cluster, within which it independently selects tasks using a decentralized strategy. This hybrid structure allows the system to retain the global efficiency of centralized planning while benefiting from the flexibility and fault tolerance of decentralized execution.

### *B. Multi-Agent Pickup and Delivery*

The Multi-Agent Pickup and Delivery (MAPD) problem involves a team of robots executing a continuous stream of pickup and delivery tasks. Each task requires transporting an object from a pickup to a delivery location. Challenges include real-time task assignment, collision-free path planning, and constraints such as capacity, heterogeneity, and time windows. Existing methods can be divided into decoupled and coupled approaches.

**Decoupled methods** separate assignment and path planning to improve scalability. Recently, the Multi-Goal MAPD (MG-MAPD) framework [44] generalized the MAPD configuration to allow robots to carry and deliver multiple items across multiple goals, where task grouping and assignment are done offline, followed by prioritized path planning. Though generally suboptimal, such decoupled strategies support large-scale and online applications with lower computational demands.

**Coupled methods** solve task assignment and path planning jointly. CBS-TA [45] modifies Conflict-Based Search to achieve optimal coordination but scales poorly. To improve performance, Chen et al. [9] proposed a marginal-cost based assignment strategy with meta-heuristic refinements, which outperforms greedy baselines under capacity constraints. These approaches improve global solution quality but are computationally intensive. Ma et al. [46] introduced Token Passing with Task Swaps (TPTS), a decentralized online method that assigns tasks greedily and uses Cooperative A* for path planning, which can scale the problems up but cannot promise the solution quality.

Our proposed method follows a decoupled framework, where task assignment and path planning are solved separately. However, unlike standard decoupled approaches, we incorporate environmental information and obstacle layouts during task clustering and allocation. This ensures that the resulting task assignments are consistent with actual path feasibility, reducing mismatch between planning stages. As a result, the approach preserves solution quality while maintaining the scalability required for large-scale scenarios.

### *C. LLM-based Robot Planning*

Recent advances in Large Language Models (LLMs) have provided new possibilities for robot planning by enabling flexible interpretation of natural language and high-level reasoning. Current work that integrates LLMs into robot planning can be grouped into two main paradigms: *LLMs as planners* and *LLMs as translators* [47].

**LLMs as planners** use LLMs directly to generate task or motion plans from natural language instructions. Early examples include Naive Task Planning [48], which outputs a full sub-task sequence without validation, and SayCan [27], which improves feasibility by scoring actions with a likelihood model. Later systems such as Text2Motion [49] and SMART-LLM [26] combine LLM reasoning with feasibility checks or multi-agent coordination. While these works highlight the potential of LLMs for high-level planning, their performance is often constrained by limited spatial grounding and execution consistency.

**LLMs as translators**, in contrast, aim to convert natural language into structured task representations that can be solved by established model-based planners. Examples include translations into PDDL [50], [51], temporal logics such as LTL or STL [24], [25], and behavior trees [52]. This paradigm offers greater stability, modularity, and compatibility with existing robotics pipelines, since the heavy lifting of planning is delegated to well-studied solvers. Recent studies have shown that translator-based pipelines often outperform planner-based

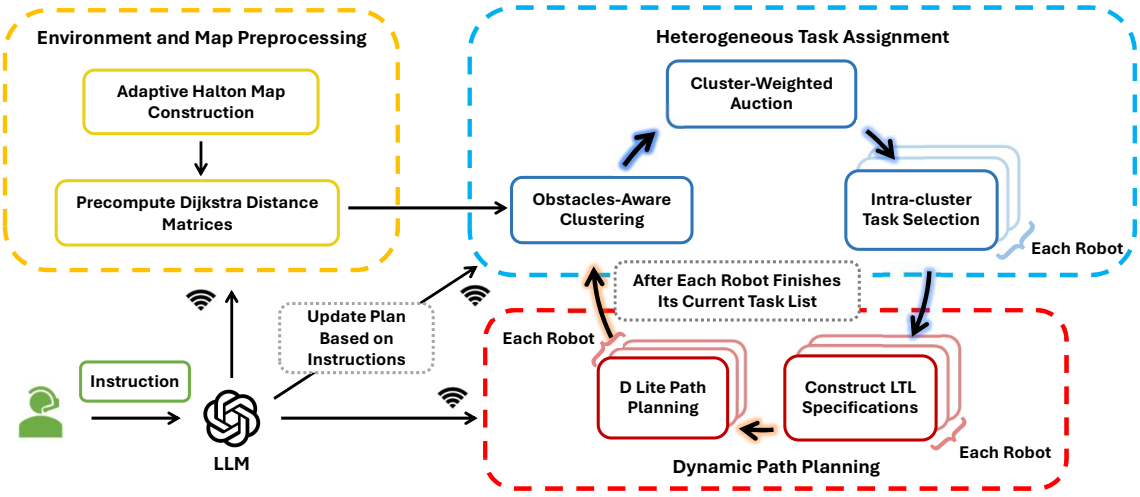



Fig. 2. Overview of the OATH framework: the planning module is composed of three parts — environment and map preprocessing, task assignment, and path planning. Task assignment and path planning operate in a closed loop, where robot plans are iteratively updated as tasks are completed. At the high level, an LLM continuously interprets human instructions, providing semantic guidance to ensure obstacle-aware and adaptive task assignment throughout the process.

ones, particularly in spatially complex environments [24]. Furthermore, vision-language models (VLMs) have been introduced to enhance perceptual grounding and bridge the gap between high-level specifications and real-world observations [53].

In our proposed method, we follow the translator paradigm but extend it to real-time use. Instead of translating instructions only once during planning, we integrate the LLM as a persistent interpreter throughout execution. When receiving a command like "There is a new task in room B; avoid the bushes near the door," the LLM identifies actionable content, constraints, and dispatches relevant information to the appropriate module. This design enables dynamic instruction handling and modular system response.

## III. Problem Formulation

We consider an MAPD problem involving multiple classes of heterogeneous robots and multiple types of tasks (see Fig. 1). Let $\mathcal{P} = \{1, 2, \ldots, P\}$ denote the set of tasks, and $\mathcal{R} = \{1, 2, \ldots, R\}$ denote the set of robots. Each task $i \in \mathcal{P}$ is associated with a pickup location $o_i$, a delivery location $d_i$, and a task type $\tau_i \in \mathcal{T}$, where $\mathcal{T}$ denotes the set of all task types and $|\mathcal{T}|$ is the total number of distinct task types.

Each robot $r \in \mathcal{R}$ is characterized by a capability vector $\boldsymbol{U}_r \in \{0,1\}^{|\mathcal{T}|}$ (or more generally $\boldsymbol{U}_r \in \mathbb{R}_{\geq 0}^{|\mathcal{T}|}$), where $(\boldsymbol{U}_r)_\tau = 1$ indicates that robot $r$ can perform task type $\tau \in \mathcal{T}$, and $(\boldsymbol{U}_r)_\tau = 0$ otherwise. This representation captures a wide range of heterogeneous team configurations by allowing robots to have distinct or overlapping capabilities across different task types.

In the meantime, each robot $r \in \mathcal{R}$ possesses a capacity $Q_r$ indicating the maximum number of tasks it can carry simultaneously. The planned execution route for robot $r$ is denoted by an ordered sequence of pickup and delivery locations, $\pi_r = [\pi_r^1, \pi_r^2, \ldots]$. Specifically, $\pi_r$ includes all pickup and delivery points assigned to robot $r$ in the current allocation round. For example, if robot $r$ is assigned to pick up items at locations $a$ and $b$ and deliver them to location $c$, then $\pi_r = [a, b, c]$ contains $a$, $b$ and $c$ in the order determined by the task assignment.

We define a binary assignment variable $\mu_{i,r} \in \{0, 1\}$, which equals 1 if task $i$ is assigned to robot $r$, and 0 otherwise. Let $a(j)$ represent the arrival time of a robot at location $j$. The function $q_r(t)$ denotes the number of tasks being carried by robot $r$ at time $t$. The travel time between any two locations $i$ and $j$ is denoted by $t(i, j)$.

**Problem 1.** Given a set of pickup and delivery tasks $\mathcal{P}$ and a heterogeneous robot team $\mathcal{R}$, the objective is to assign each task $i \in \mathcal{P}$ to a robot $r \in \mathcal{R}$ and plan a route $\pi_r$ for each robot to find an optimal solution, i.e., minimize the total travel cost.

The travel cost for robot $r$ is defined as the total travel time along its planned route, considering its motion and capability constraints:

$$\sum_{j=1}^{|\pi_r|-1} t(\pi_r^j, \pi_r^{j+1}) \tag{1}$$

The overall objective is to minimize the total travel cost of all robots:

$$\min \quad \sum_{r \in \mathcal{R}} \sum_{j=1}^{|\pi_r|-1} t(\pi_r^j, \pi_r^{j+1}) \tag{2a}$$

$$\text{s.t.} \quad \mathcal{P}_r \cap \mathcal{P}_{r'} = \emptyset, \qquad \forall r, r' \in \mathcal{R},\ r \neq r', \tag{2b}$$

$$\bigcup_{r \in \mathcal{R}} \mathcal{P}_r = \mathcal{P}, \tag{2c}$$

$$q_r(t) \leq Q_r, \qquad \forall r \in \mathcal{R},\ \forall t, \tag{2d}$$

$$\mu_{i,r} = 0, \qquad \text{if } (\boldsymbol{U}_r)_{\tau_i} = 0,\ \forall i \in \mathcal{P},\ \forall r \in \mathcal{R}, \tag{2e}$$

$$a(o_i) < a(d_i), \qquad \forall i \in \mathcal{P}, \tag{2f}$$

$$\mu_{ir} = 1 \Rightarrow o_i, d_i \in \pi_r, \forall i \in \mathcal{P},\ \forall r \in \mathcal{R}. \tag{2g}$$

Constraint (2b) demands that each task is assigned to exactly one robot, and Constraint (2c) guarantees full task coverage.

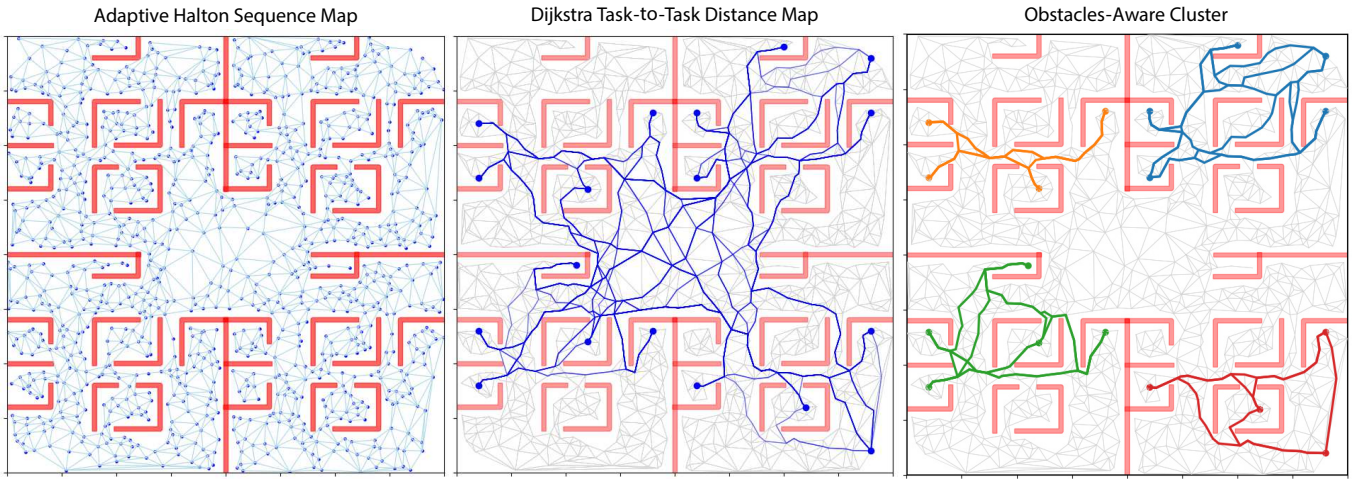



Fig. 3. Spatial modeling and task clustering process. The left figure shows the construction of the adaptive Halton sequence, generating obstacle-aware sampling points. The middle figure presents the Dijkstra result computed over the Halton map, producing a task-to-task distance matrix. The right figure visualizes the final clustering result: four clusters are generated using agglomerative clustering [54] based on the distance matrix, where each color represents one cluster. Notably, the clustering respects the environmental complexity, avoiding grouping tasks across walls or obstacle-dense regions.

Constraint (2d) enforces the capacity limit of each robot. Constraint (2e) generalizes the type compatibility rule: a robot $r$ can only be assigned tasks whose type $\tau_i$ satisfies $(\boldsymbol{U}_r)_{\tau_i} = 1$, i.e., the robot is capable of executing that task type. Constraint (2f) imposes pickup-before-delivery ordering for each task, and Constraint (2g) requires that both pickup and delivery locations of any assigned task appear in the corresponding robot's route.

Due to the NP-hard nature [1] of Problem 1, finding an optimal solution is computationally expensive, particularly for large-scale scenarios. To address this challenge, we scaffold an efficient and modular planning framework, as illustrated in Fig. 2, which incorporates a high-level planner for task assignment and low-level planner for path routing. In addition, we integrate it with a human-in-the-loop interface powered by an LLM for enhanced adaptability and interaction.

## IV. Methods

Many existing approaches for MATP perform task grouping prior to assignment, but typically rely on geometric proximity (e.g., Euclidean distance) without explicitly accounting for environmental obstacles [14], [19], [55]. In cluttered or complex environments, this often leads to inaccurate task groupings and, consequently, suboptimal task assignments and inefficient path planning.

To address this limitation, we introduce obstacle awareness into the task assignment pipeline through two key strategies. First, we construct an adaptive Halton sequence map that generates sampling points whose density varies according to the obstacle distribution. Second, we apply a Dijkstra algorithm to compute a task-to-task distance matrix that accurately reflects traversability under obstacle constraints.

Based on this environment representation, our obstacle-aware task assignment and planning framework operates fully online and adapts to heterogeneous robot teams and dynamic environments. The process begins with a cluster-based auction mechanism that assigns groups of tasks to individual robots. Each robot then performs intra-cluster task selection with capacity constraints and generates a suboptimal global path to execute its assigned tasks. Upon completing a task set, the robot re-enters the assignment and planning cycle. This iterative process continues until all tasks are completed.

An overview of the full pipeline is illustrated in Fig. 2, and the major components are detailed in the following subsections. We also provide a formal asymptotic complexity analysis to characterize how the computational cost scales with problem size in Appendix A.

### A. Environment and Map Preprocessing

The environment and map preprocessing stage consists of two main components: (1) adaptive Halton sequence map construction, and (2) Dijkstra distance matrix computation.

*1) Adaptive Halton Sequence Map:* To provide a flexible and obstacle-aware spatial foundation for both task assignment and path planning, we adopt an *adaptive Halton sequence map* instead of a conventional grid-based representation.

A Halton sequence $\{h_i\}_{i=1}^{N} \subset [0,1]^{\epsilon}$ is an $\epsilon$-dimensional low-discrepancy sequence built from pairwise coprime bases $b_1, \ldots, b_\epsilon$. For each point index $i \in \{1, \ldots, N\}$, write

$$h_i = \left(h_i^{(1)}, \ldots, h_i^{(\epsilon)}\right),$$

where $h_i^{(j)}$ is the $j$-th coordinate of the $i$-th point, with $j \in \{1, \ldots, \epsilon\}$. For each $j$, this coordinate is given by the radical inverse in base $b_j$:

$$h_i^{(j)} = \phi_{b_j}(i) = \sum_{k=0}^{L} \frac{\alpha_k}{b_j^{k+1}}, \quad i = \sum_{k=0}^{L} \alpha_k b_j^k \tag{3}$$

where $\alpha_k$ are the digits of $i$ in base $b_j$. The resulting sequence fills the space $[0,1]^{\epsilon}$ more evenly than pseudo-random points.

Building on the Halton sequence, we construct an adaptive Halton sequence map by introducing an acceptance probability

function that determines whether a candidate point is sampled at a given location. The acceptance probability for a candidate point is defined as:

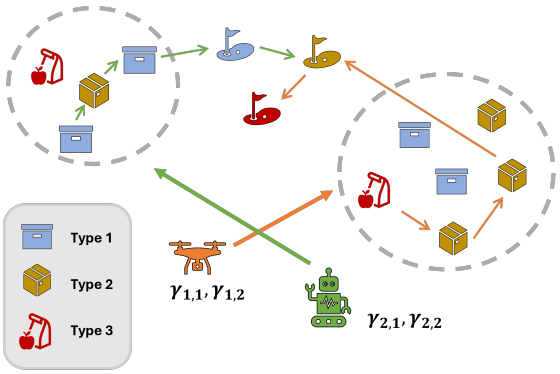
$$P_{\text{accept}}(\delta) = \begin{cases} 0, & \delta < \delta_{\min} \\ \beta + (1-\beta)\exp\left(-\frac{(\delta-\delta_{\text{opt}})^2}{2\sigma^2}\right), & \delta \geq \delta_{\min} \end{cases} \tag{4}$$

where $\delta$ is the distance from the candidate point to the nearest obstacle, $\delta_{\min}$ is the minimum safe distance, and $\delta_{\text{opt}}$ is the optimal distance for sampling where the acceptance probability is highest. The parameter $\sigma$ controls the width of the Gaussian distribution, and $\beta \in [0, 1]$ represents the minimum acceptance probability (i.e., the baseline floor value) in moderately constrained areas.

For each Halton-generated candidate point, we draw a random variable $u \in [0, 1]$. The point is accepted if it satisfies the condition $u < P_{\text{accept}}(d)$. This process results in a non-uniform, obstacle-aware distribution of sampling points that is used consistently across task preprocessing and path planning.

The left image in Fig. 3 shows the result of this adaptive sampling process. Compared to a regular grid that produces a fixed lattice, the adaptive Halton sequence provides non-uniform yet well-distributed sampling, offering smoother spatial coverage and improved connectivity that naturally forms a triangular navigation structure rather than axis-aligned grid edges. When applied in our setting, these properties yield three key advantages over grid maps with the same number of sampled points: (1) the adaptive Halton map offers more accessible entries to narrow passages, increasing the likelihood of finding alternate routes in multi-robot planning; (2) paths generated on the Halton map are typically smoother and shorter as they avoid the fixed $90°$ or $45°$ turns inherent in grid-based maps; and (3) the adaptive Halton sampling can dynamically adjust point density to local obstacle distributions, a flexibility that fixed grid representations lack.

*Remark* 1 (Practical Parameter Tuning). The parameters in the probabilistic acceptance function of Eq. 4 are tunable and have intuitive geometric interpretations. The threshold $\delta_{\min}$ enforces a safety margin from obstacles, while $\delta_{\text{opt}}$ specifies the distance at which sampling is most informative for navigation. The parameters $\sigma$ and $\beta$ control the smoothness of density decay and the minimum sampling density in open regions, respectively. In practice, these parameters are adjusted based on external factors such as the map scale, obstacle density, and robot footprint. A typical tuning procedure is as follows. First, $\delta_{\min}$ is set according to the robot size and safety margin to ensure collision-free sampling. Next, $\delta_{\text{opt}}$ is chosen slightly larger than $\delta_{\min}$ to promote denser sampling near obstacle boundaries, which are critical for feasible navigation. The parameter $\sigma$ is then used to regulate how broadly this dense sampling region extends, with larger values producing smoother density transitions and smaller values emphasizing narrow passages. Finally, $\beta$ is selected to guarantee a minimum sampling density in open regions. The tuning objective is to ensure that all task locations remain mutually reachable in the resulting roadmap, while narrow passages are sufficiently sampled to admit multiple feasible traversal paths. Once this connectivity criterion is satisfied, further adjustment of parameters has limited impact on downstream task allocation performance. We empirically evaluated the effect of varying $\delta_{\text{opt}}$, $\sigma$, and $\beta$ within practical ranges. Across a wide range of parameter combinations that satisfy the above connectivity objective, we observe similar roadmap connectivity and comparable task allocation performance, which indicates that the method is not sensitive to precise parameter values.




| Bidding Scores | Cluster 1 | Cluster 2 |
|---|---|---|
| Drone 1 | $Score_{1,1}/\gamma_{1,1}$ | $Score_{1,2}/\gamma_{1,2}$ |
| Ground robot 2 | $Score_{2,1}/\gamma_{2,1}$ | $Score_{2,2}/\gamma_{2,2}$ |

Fig. 4. Illustration of the hierarchical task-assignment framework. Tasks are clustered using spatial proximity and obstacle-aware distances. A cluster-level auction assigns task groups to robots based on robot capabilities and the cluster's task-type distribution, denoted by $\gamma$. Within each assigned cluster, each robot selects tasks subject to capacity limits and task-type compatibility.

*2) Dijkstra-based Distance Matrix:* In our framework, we apply a Dijkstra algorithm to the previously constructed adaptive Halton sequence map. Specifically, each task location, both pickup and delivery points, is treated as a source node, and shortest paths to all other task locations are calculated in advance. This produces a task-to-task distance matrix

$$\begin{aligned} &\mathcal{M} = [m_{ij}], \quad m_{ij} = \delta(v_i, v_j), \\ &v_i, v_j \in \{o_1, \ldots, o_n, d_1, \ldots, d_n\}. \end{aligned} \tag{5}$$

that accurately reflects real-world traversability by taking into account obstacle constraints, where $\delta(\cdot, \cdot)$ denotes the obstacle-aware shortest-path distance.

By aligning the clustering phase with the same spatial model used in subsequent path planning (e.g., D*-Lite), our approach ensures consistency across the system. An example of the resulting Dijkstra output is shown in the middle image of Fig. 3. This graph captures obstacle-aware traversal costs and is used in the following clustering and allocation modules in Sec. IV-B.

### B. Heterogeneous Task Assignment

To address the challenges of task assignment in a heterogeneous robot team with capacity constraints, we propose a novel hierarchical task assignment framework. The overall process operates iteratively as shown by the loop in Fig. 2: in each round, all currently unassigned tasks are clustered based on the previously computed task-to-task distance matrix; robots then participate in a cluster-level auction, select tasks within

**Algorithm 1:** Cluster statistics & robot–cluster scoring

**Input:** Unassigned task set $\mathcal{P}_u$; task-to-task distance matrix $\mathcal{M}$; robot set $\mathcal{R}$; number of task types $|\mathcal{T}|$; robot capability vectors $\{\boldsymbol{U}_r \in \mathbb{R}^{|\mathcal{T}|}_{\geq 0}\}$
**Output:** Clusters $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$; robot-specialization scores $\{\gamma_{r,k}\}$; score matrix $S = \{s_{r,k}\}$

1 **Clustering:** Cluster all tasks $p \in \mathcal{P}_u$ using agglomerative clustering on $\mathcal{M}$ into clusters $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$;
2 **Cluster statistics: foreach** *cluster* $\mathcal{C}_k$ **do**
3 Compute type-count vector $\boldsymbol{N}_k \in \mathbb{R}^{|\mathcal{T}|}_{\geq 0}$;
4 Define cluster composition: $\psi_k \leftarrow \dfrac{\boldsymbol{N}_k}{\|\boldsymbol{N}_k\|_1}$;
5 **Robot preferences: foreach** *robot* $r \in \mathcal{R}$ **do**
6 Normalize capability: $\zeta_r \leftarrow \dfrac{\boldsymbol{U}_r}{\|\boldsymbol{U}_r\|_1}$;
7 **Scoring: foreach** *robot* $r \in \mathcal{R}$ **do**
8 **foreach** *cluster* $\mathcal{C}_k$ **do**
9 Compute distance $\delta_{r,k}$ from $r$ to the center of $\mathcal{C}_k$;
10 Compute robot-specialization score: $\gamma_{r,k} \leftarrow \langle \psi_k, \zeta_r \rangle \in (0, 1]$;
11 Set score: $s_{r,k} \leftarrow \dfrac{\delta_{r,k}}{\gamma_{r,k}}$;
12 **return** $\{\mathcal{C}_k\}$, $\{\gamma_{r,k}\}$, *and* $S = \{s_{r,k}\}$

their assigned clusters[1], and execute them. After completing the current assigned tasks, the robots re-enter the assignment cycle, where the remaining unassigned tasks are clustered again and new intra-cluster task selections are performed. This iterative loop continues until all tasks have been successfully completed. The full procedure consists of three main components: Obstacle-Aware Clustering, Cluster-Weighted Auction, and Intra-Cluster Task Selection. An illustration of this hierarchical task-assignment framework is shown in Fig. 4. The following sections describe each stage in detail.

*1) Obstacle-Aware Clustering:* Given the task-to-task distance matrix from Sec. IV-A2, we group tasks using agglomerative clustering [54]. In Algorithm 1, Lines 1-4 correspond to this clustering process. Unassigned tasks are divided into clusters $\{\mathcal{C}_1, \ldots, \mathcal{C}_K\}$ based on the task-to-task distance matrix $\mathcal{M}$.

In general, the environment may contain multiple task types $\tau_i$. Each cluster $\mathcal{C}_k$ can thus be characterized by the distribution of task types it contains. Let $\boldsymbol{N}_k \in \mathbb{R}^{|\mathcal{T}|}_{\geq 0}$ denote the vector of task counts of each type within $\mathcal{C}_k$. To describe the normalized composition of task types in the cluster, we define

$$\psi_k \;=\; \frac{\boldsymbol{N}_k}{\|\boldsymbol{N}_k\|_1} \tag{6}$$

To prevent division by zero during the subsequent scoring calculation, a small constant $\theta$ is added to $\boldsymbol{N}_k$ in the equation above during our implementation. This constant is sufficiently small to minimize its effect on the task assignment results.

[1]Note that, the robot usually will select only a subset of the total tasks within the assigned cluster, due to the limited capacity of the robot.

**Algorithm 2:** Cluster-Weighted Auction (Assignment Stage)

**Input:** Robot set $\mathcal{R}$; clusters $\{\mathcal{C}_1, \ldots, \mathcal{C}_K\}$; score matrix $S = \{s_{r,k}\}$
**Output:** Cluster assignment set $\mathcal{A}$

Initialize best-bid record $B[k] \leftarrow \infty$ for all $k$, and $\mathcal{A} \leftarrow \emptyset$;
**repeat**
**foreach** *robot* $r \in \mathcal{R}$ **do**
Select cluster index $k^\star \leftarrow \arg\min_k \{\, s_{r,k} \mid s_{r,k} < B[k]$ and $\mathcal{C}_k$ available $\}$;
**if** $k^\star$ *exists* **then**
$B[k^\star] \leftarrow s_{rk^\star}$; assign $r \rightarrow \mathcal{C}_{k^\star}$;
Resolve conflicts if any and update $\mathcal{A}$;
**until** *all robots have conflict-free cluster assignments*;
**return** $\mathcal{A}$

This normalized vector $\psi_k$ captures the relative proportion of each task type within the cluster and will be leveraged in the subsequent auction step for scoring. In the next stage, $\psi_k$ is inner-producted with each robot's capability vector to compute a robot–cluster specialization score that reflects how well a robot matches the task composition of $\mathcal{C}_k$.

*Remark* 2 (Obstacle-Aware Distance and Clustering Choice). The agglomerative clustering is adopted because it supports arbitrary, non-Euclidean distance metrics, making it suitable for obstacle-rich environments where straight-line measures are often misleading. For instance, $k$-means clustering relies on Euclidean or Manhattan distances, which treat two tasks separated by a wall as "close," even though the feasible traversable path in reality is actually long. By using obstacle-aware distances computed via Dijkstra on the adaptive Halton map, our method avoids such idealistic groupings.

*Remark* 3 (Selection and Sensitivity of Cluster Number). The number of clusters can be determined using an elbow-style analysis based on within-cluster dispersion, as illustrated in the upper subplot of Fig. 5. In this setting, the within-cluster sum of squares (WCSS) curve exhibits a clear elbow at $K = 4$, indicating that further increasing the number of clusters leads to diminishing reductions in within-cluster dispersion. Therefore, $K = 4$ provides an effective trade-off between cluster compactness and task allocation overhead, and is selected as the default cluster number for the experiments conducted in the same map shown in Fig. 1. Specifically, we define the sensitivity of a performance metric $f(K)$ as

$$\text{Sensitivity}(f) = \frac{\max_K f(K) - \min_K f(K)}{\frac{1}{|\mathcal{K}|}\sum_{K \in \mathcal{K}} f(K)},$$

where $\mathcal{K} = \{4, 5, 6, 7, 8\}$. Based on this definition, the sensitivity of the total completion time is approximately $7.8\%$, and the sensitivity of the total number of steps is approximately $8.2\%$. These results indicate that OATH is not highly sensitive to the choice of the cluster number within a reasonable range.

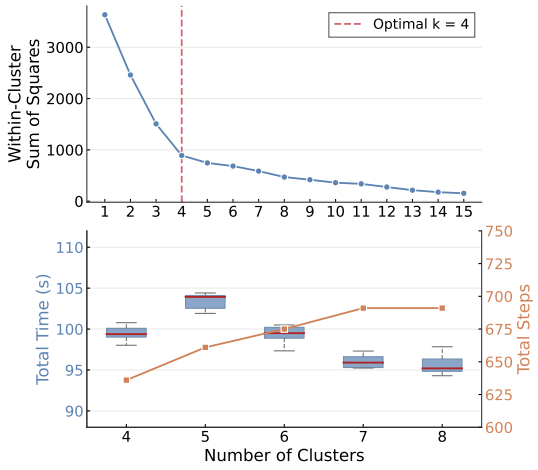



Fig. 5. **Top:** Elbow-style analysis using the within-cluster sum of squares (WCSS), indicating $K = 4$ as a reasonable choice. **Bottom:** Total completion time (boxplots) and total number of steps (line) under different cluster numbers, showing limited performance variation within a reasonable range of $K$.

In practice, after selecting an appropriate cluster number, we additionally ensure that the number of clusters is slightly larger than (or at least equal to) the number of robots. This design increases the flexibility of the cluster–auction process and promotes more balanced and localized task groupings across robots. Accordingly, the algorithm enforces that the number of clusters is greater than the number of robots, except when the total number of tasks is smaller than the number of robots. In that case, the number of clusters is set equal to the number of tasks to ensure full task coverage without redundant clustering.

*2) Cluster-Weighted Auction (CWA):* The CWA assigns clusters of tasks to robots by jointly considering spatial proximity and task-type compatibility. After task clustering, CWA assigns each robot to a cluster where it is both geographically close (in obstacle-aware distance) and well aligned with the cluster's task composition. This process consists of two stages: (i) score computation and (ii) the auction phase.

For score computation phase, each robot $r \in \mathcal{R}$ is represented by a capability vector $\boldsymbol{U}_r \in \mathbb{R}_{\geq 0}^{|\mathcal{T}|}$, We define the normalized preference of each robot as

$$\zeta_r = \frac{\boldsymbol{U}_r}{\|\boldsymbol{U}_r\|_1}. \tag{7}$$

For each cluster $\mathcal{C}_k$ with its task-type composition vector $\psi_k$, the compatibility between robot $r$ and cluster $\mathcal{C}_k$ is quantified by a robot–cluster specialization factor:

$$\gamma_{r,k} = \langle \psi_k, \zeta_r \rangle \in (0, 1], \tag{8}$$

which measures how well the robot's capability distribution aligns with the type composition of the cluster. Here, $\langle \cdot, \cdot \rangle$ denotes the inner product between the two normalized vectors $\psi_k$ and $\zeta_r$.

Each robot then estimates its distance $\delta_{r,k}$ to the cluster center, and combines it with $\gamma_{r,k}$ to form the bidding score:

$$\text{score}_{r,k} = \frac{\delta_{r,k}}{\gamma_{r,k}}, \tag{9}$$

such that robots closer to the cluster and better matched to its task distribution have a higher probability to win the auction. Algorithm 1 (Lines 5-12) implements this scoring step.

Once all scores are calculated, the auction proceeds in a sequential manner. Robots are ordered (e.g., by robot index number), and each robot selects the cluster with the best (lowest) score that is still available. If a robot selects a cluster that has already been claimed by another robot, the previous assignment is revoked, and that robot will re-enter the auction in the next iteration. Algorithm 2 implements this auction step.

This bidding and conflict resolution process continues until all robots are assigned to non-overlapping clusters. The final assignment result is stored in the cluster assignment set $\mathcal{A}$, which maps each robot to a task cluster for subsequent intra-cluster planning.

*3) Intra-Cluster Task Selection:* After tasks have been grouped into smaller clusters and assigned to individual robots, each robot selects a subset of type-compatible tasks from its assigned cluster, within its capacity limit. The goal is to determine the locally optimal sequence of pickup and delivery locations that minimizes travel cost. This process can be formulated as a small-scale Traveling Salesman Problem (TSP) with constraints, which is efficiently solvable as MILP using an off-the-shelf optimization solver (e.g., Gurobi). The formal problem definition is given below.

**Problem 2.** Given a robot $r$ with capacity $Q_r$ and an assigned task set $\mathcal{C}_r = \{1, 2, \ldots, n_r\}$, where $n_r$ is the number of tasks in the cluster, the objective is to determine the optimal pickup–delivery sequence over obstacle-aware locations that minimizes total travel cost, subject to task type compatibility, capacity, and precedence constraints.

Let $\mathcal{V}_r = \{v_1, v_2, \ldots, v_{n_r}, \ldots, v_{n_r+m_r}\}$ denote the set of all pickup and delivery locations associated with tasks in $\mathcal{C}_r$, where $n_r$ is the number of pickups (equal to the number of tasks) and $m_r$ is the number of deliveries. Define $x_{ij} \in \{0, 1\}$ as a binary variable indicating whether the robot travels directly from $v_i$ to $v_j$, $\ell_i$ as the carried load upon arrival at $v_i$, and $a_i$ as the arrival time at $v_i$. $u_i$ is an integer ordering variable associated with location $v_i$, indicating its position in the tour (i.e., the visit order); and $\delta(i, j)$ is the obstacle-aware travel cost from location $v_i$ to $v_j$, defined previously in Sec. IV-A2.

Intra-cluster task selection for robot $r$ is formulated using MILP:

$$\min_{x} \sum_{v_i \in \mathcal{V}_r} \sum_{v_j \in \mathcal{V}_r} \delta(i, j)\, x_{ij} \tag{10a}$$

$$\text{s.t.} \sum_{v_j \in \mathcal{V}_r} x_{ij} = 1, \qquad \forall v_i \in \mathcal{V}_r \tag{10b}$$

$$\sum_{v_i \in \mathcal{V}_r} x_{ij} = 1, \qquad \forall v_j \in \mathcal{V}_r \tag{10c}$$

$$0 \leq \ell_i \leq Q_r, \qquad \forall v_i \in \mathcal{V}_r \tag{10d}$$

$$a_{p_i} < a_{d_i}, \qquad \forall v_i \in \mathcal{V}_r \tag{10e}$$

$$x_{ij} = 0, \qquad \text{if } (\boldsymbol{U}_r)_{\tau_{i/j}} = 0 \tag{10f}$$

$$u_i + 1 \leq u_j + |\mathcal{V}_r|\,(1 - x_{ij}), \forall i \neq j,\ v_i, v_j \in \mathcal{V}_r \tag{10g}$$

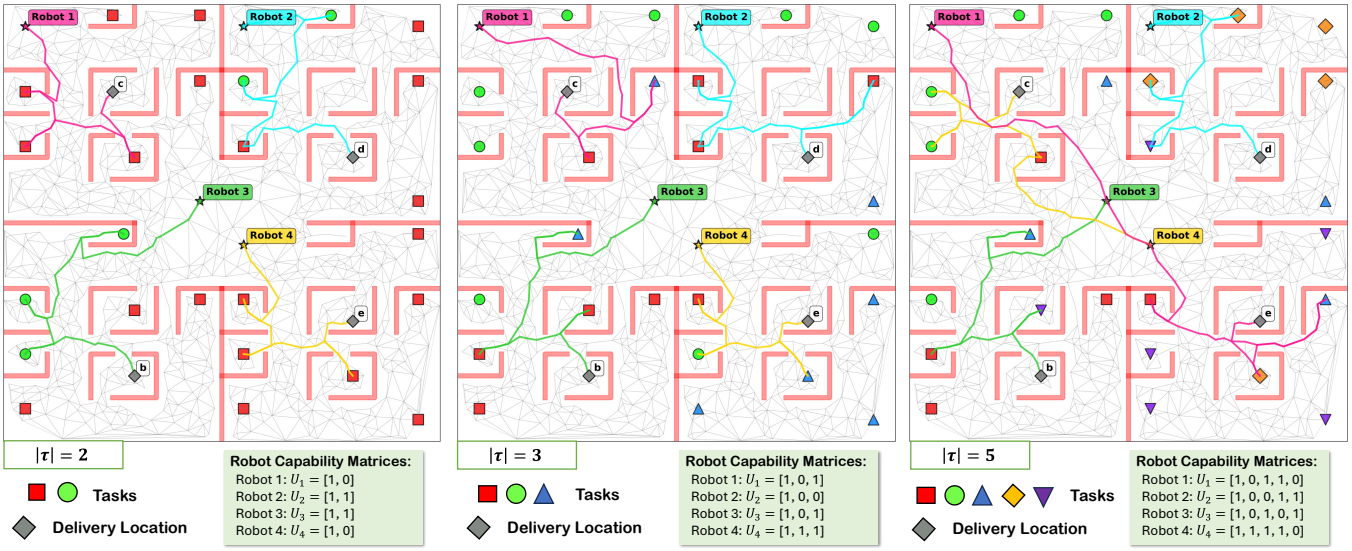



Fig. 6. Comparison of heterogeneous task assignment results under different numbers of task types. Each subfigure illustrates the final task assignment for teams operating with $|\mathcal{T}| = 2, 3, 5$, respectively. The results show that each robot strictly follows its capability matrix when selecting tasks. Even when a nearby cluster contains incompatible task types, the robot does not choose it but instead travels to a more distant cluster whose tasks match its capabilities (e.g., in the $|\mathcal{T}| = 5$ case, Robot 1 selects a farther cluster that aligns with its feasible task set).

Constraints (10b) and (10c) ensure that each location is visited exactly once. Constraint (10d) imposes the robot's capacity limit $Q_r$; Constraint (10e) guarantees that each pickup precedes its corresponding delivery; Constraint (10f) eliminates infeasible assignments based on robot–task compatibility; and Constraint (10g) eliminates subtours and ensures route connectivity. We adopt the Miller–Tucker–Zemlin (MTZ) formulation [56], a classical linearization technique widely used in TSP/VRP problems. In the MTZ formulation, the auxiliary variables $u_i$ represent the visiting order of each location, same as we do. Specifically, Constraint (10g) enforces that if edge $(i, j)$ is included in the route ($x_{ij} = 1$), then node $j$ must appear after node $i$ in the visiting sequence; when $x_{ij} = 0$, the inequality is relaxed by the large constant term $|\mathcal{V}_r|$. Together, this constraint prevents the formation of disconnected loops and guarantees a single continuous tour for each robot.

The results of the heterogeneous, obstacle-aware task assignment are illustrated in Fig. 6. The figure presents three allocation outcomes under different numbers of task types, $|\mathcal{T}| = 2, 3$, and 5, respectively. Each robot is assigned to a cluster of tasks that are both spatially accessible in free space (i.e., regions without long wall segments) and compatible with the robot's capability vector. In particular, when the nearest cluster contains incompatible task types, the robot selects a more distant but feasible cluster instead, as observed in the $|\mathcal{T}| = 5$ case where Robot 1 chooses a further cluster that aligns with its task-handling abilities. These results demonstrate that the proposed heterogeneous task assignment algorithm effectively balances spatial proximity and capability matching, enabling each robot to select reachable and compatible task clusters while maintaining obstacle-aware and efficient allocations.

*Remark* 4 (Global Optimality and Replanning Trade-off). Within each cluster, the MILP produces an optimal sequence for the tasks assigned to a single robot under the current cluster composition and capacity constraints. The overall framework, however, does not guarantee global optimality: cluster assignments and routes are determined iteratively and may become suboptimal as execution unfolds. Increasing the frequency of the cluster–auction–intra-cluster selection cycle can improve realized path quality by re-optimizing more often, but it also incurs additional task reallocation and replanning overhead, which may increase wall-clock mission time. In practice, there is a trade-off between solution quality and computational efficiency.

*Remark* 5 (Motion Heterogeneity). Beyond the binary capability vector $U_r$, OATH supports motion heterogeneity through robot-specific transition systems. During roadmap construction, we define robot-specific feasibility constraints and edge costs. Differences in obstacle-surmounting ability are encoded either by adjusting edge weights or by removing infeasible transitions. Since allocation relies on robot-specific shortest-path distances, these differences naturally propagate to cluster formation and route selection without modifying the core framework.

### C. Dynamic Path Planning

Once each agent has received its individual task assignment, it must execute the assigned tasks in a specific order while avoiding unintended goal locations. To enforce temporal constraints and ensure correctness of task execution, we employ Linear Temporal Logic (LTL) to formally encode the required task sequences for each agent. LTL provides a compact language for specifying ordering and safety requirements, such as "task $A$ must eventually follow task $B$" or "task $C$ must never be visited before task $D$."

For each agent, we automatically construct an LTL formula based on its assigned tasks. This formula is then translated into

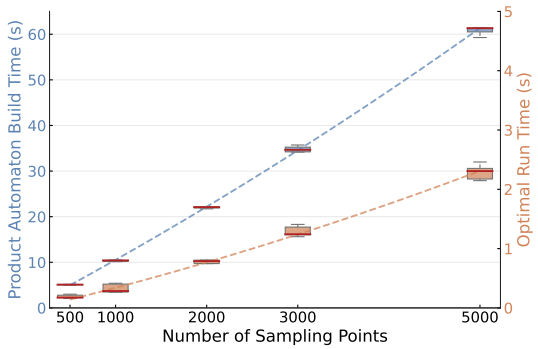



Fig. 7. Sensitivity analysis of product automaton construction time and optimal-run search time with respect to the number of sampling points. Blue boxplots represent the product automaton construction time, and orange boxplots represent the optimal-run computation time. For each sampling density, the experiment is repeated five times using the same LTL specification but different random seeds, resulting in distinct sampling configurations. Results are summarized using boxplots.

a Büchi automaton [57], a finite-state machine that accepts infinite sequences satisfying the LTL specification. The Büchi automaton is then composed with a transition system that models the environment, yielding a product automaton. The product automaton captures both the temporal requirements and the spatial structure of the workspace, ensuring that any path found is correct with respect to the assigned specification.

Path planning is then carried out on the product automaton using the D*-Lite algorithm [58], a dynamic and heuristic search method that supports efficient replanning. This allows agents to update their paths during execution in response to newly detected or previously unknown obstacles, maintaining the execution feasibility while adapting to changes in the environment.

*Remark* 6 (Product Automaton Construction Cost). The time required to construct the product automaton depends on two main factors. First, the complexity of the LTL formula increases with the number of tasks encoded; for example, planning a route for ten tasks results in a significantly longer specification than planning for only three. Second, the size of the environment model which is determined by the number of sampled points in the underlying map, directly affects the number of states in the transition system and thus the computational cost of constructing the product automaton. In our case, sampling $1000$ adaptive Halton points generates a transition system that is combined with a Büchi automaton containing $5$ states and $12$ transitions. The resulting product automaton used for path planning has about $20,520$ states and $294,792$ transitions. If the LTL specification remains the same but the number of sampled points increases to $2000$, the resulting product automaton expands to about $40,520$ states and $613,800$ transitions, causing the construction time to roughly double. To further illustrate the time consumption of product automaton construction and optimal-run search under increasing sampling densities, Fig. 7 reports a sensitivity analysis. The blue boxplots shows the time required to build the product automaton, while the orange boxplots shows the time required to find an optimal run. As the number of sampling points increases, both components exhibit a clear increasing trend, with the construction of the product automaton dominating the overall runtime. Since each agent must rebuild its product automaton after every task assignment cycle, this construction cost is a critical factor. Our use of the adaptive Halton sequence provides an additional advantage: in cluttered regions, higher sampling density captures obstacle geometry sufficiently to guarantee feasible paths, while in open areas, points are sampled at lower density and in fewer numbers. By reducing unnecessary points in open regions, the adaptive map helps control the size of the product automaton and thus shortens automaton synthesis time. Although incremental construction of the product automaton is desirable in general, it is not directly applicable in our task allocation setting. In OATH, task allocation and path planning are performed in an iterative closed loop, where each allocation round produces a new LTL specification for each robot based on its updated task sequence. As a result, the Büchi automaton itself changes across allocation cycles, which precludes reuse or incremental update of a previously constructed product automaton. Nevertheless, incremental updates are exploited whenever the task-level specification remains unchanged. In particular, during execution-level replanning triggered by environmental changes (e.g., newly detected obstacles), the Büchi automaton is fixed, and replanning is performed by locally modifying the product automaton through edge invalidation and incremental cost repair using D*-Lite. This design ensures that full Büchi–product automaton reconstruction is performed only when task-level temporal specifications change, while incremental updates are used whenever possible to reduce replanning overhead.

## V. Large Language Model as Translator

Different from previous methods that only use large language models (LLMs) as an offline planner or as a translator in the pre-execution stage, our framework integrates the LLM as a persistent interpreter throughout the whole task execution process. In this way, the LLM becomes an always-available interface between human instructions and the planning system.

As shown in Fig. 8, the interaction between the LLM and the planner follows a clear pipeline. After receiving a new instruction, the LLM first determines the intent (e.g., adding a task, updating an obstacle or changing task priority) and extracts key parameters such as positions, labels, or constraints. These parameters are converted into a JSON structure and passed to the high-level planner. The planner then updates the map and the task graph according to a predefined pipeline.

In detail, the instruction “We have got a new special task showing up at coordinates five and nineteen point five. Take it over to point C.” is recognized as an *Add New Task* command. The LLM extracts the task’s pickup coordinates $(5, 19.5)$, the delivery point $C$, and auxiliary fields such as task type. Rather than rebuilding the entire map, the planner performs an incremental update: it keeps the existing adaptive Halton sequence roadmap, inserts a new task node at $(5, 19.5)$, and uses Delaunay triangulation (i.e., connecting the node to its nearest spatial neighbors while preserving non-overlapping edges) to maintain geometric consistency. The new task node is then

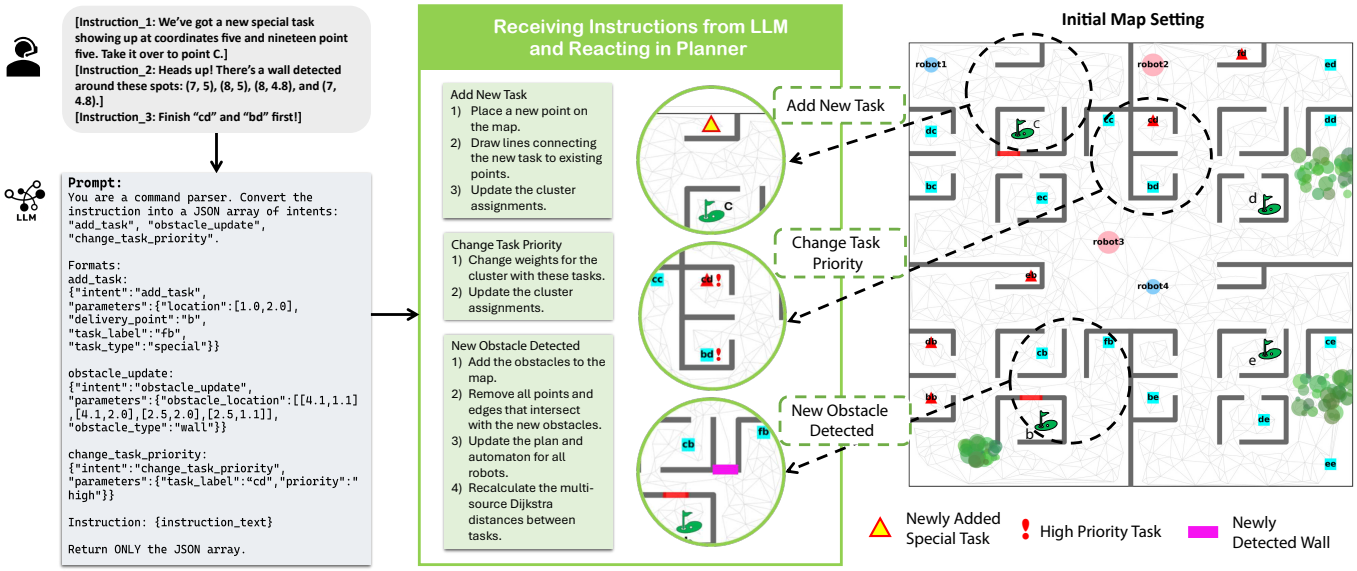



Fig. 8. Overview of how the LLM parses human instructions and how the planner reacts. The LLM converts natural language into structured intents (`add_task`, `obstacle_update`, `change_task_priority`). The planner then updates the map and task assignment accordingly (center panel). The right panel shows an example environment with 4 robots (two drones and two ground robots), delivery tasks (blue squares), inspection tasks (red triangles), delivery points (green flags), and bushes (green circles). Letter pairs shown next to tasks (e.g., `dc`, `ec`, `bb`) are task labels used by the planner and in instructions.

incorporated in the next cluster–auction–intra-cluster cycle; when robots rebuild their product automata, they automatically integrate the new coordinates, while robots currently executing tasks remain unaffected. This update process typically completes within about five seconds in our simulation setup. This 5-second delay refers to the end-to-end system response time rather than the LLM inference time alone. The total response time includes roadmap update, distance matrix computation, clustering, task reassignment, and path replanning.

For a constraint-oriented instruction such as "Heads up! There's a wall detected around these spots: $(7, 5)$, $(8, 5)$, $(8, 4.8)$, and $(7, 4.8)$," the LLM interprets it as an *New Obstacle Detected* command. The detected obstacle coordinates are passed to the planner, which removes affected nodes and edges, updates the map, and replans the route to avoid the new wall.

Similarly, for an instruction *Change Task Priority*, the planner adjusts the weight of the affected cluster and updates the corresponding task assignments. The planner's response and its impact on robot routes are illustrated in Fig. 9. The top row shows the original task assignments and planned routes, while the bottom row depicts the updated trajectories after executing the new LLM-parsed instructions. In the *Add New Task* scenario, the ground robot is initially assigned to a nearby task. When the new special task appears, which can only be handled by the ground robot, the system immediately reassigns it to complete this task first, even though it is farther away. In the *New Obstacle Detected* case, the drone dynamically adjusts its trajectory to circumvent the obstacle without redundant exploration. In the *Change Task Priority* scenario, when a distant task is assigned a higher priority, the ground robot modifies its plan and proceeds to complete that task first, reflecting an immediate reordering of objectives.

Overall, these examples demonstrate that the LLM-guided planner can seamlessly integrate new high-level instructions into ongoing task assignment and path planning. Human operators can intervene at any time without interrupting the execution process. This allows language-driven updates to tasks, environments, and constraints to be processed quickly and flexibly, which is crucial for planning in dynamic and uncertain environments.

To support more natural and less structured user inputs, we design an interactive human–LLM command interface. Rather than requiring fully specified commands, the interface allows users to issue high-level or ambiguous instructions and engage in a short clarification dialogue with the LLM. The LLM parses the intent of each instruction, identifies any parameters required by the planner that are missing or ambiguous, and explicitly requests additional information from the user when needed. The interface illustration and the complete prompt details are provided in Appendix B.

## VI. Experiments

We evaluate OATH on obstacle-rich maps (Fig. 1) containing both known walls and unknown obstacles (e.g., iron gates and bushes). An adaptive Halton sequence is generated to capture the map structure and provide sampling points. We compare our method OATH against three alternatives, including two ablation experiments and a CBBA [10] baseline. All methods use the same map instantiation, robot team, and task set, and rely on LTL-D* [58] for path planning, so differences arise solely from task assignment algorithm. We verify the proposed framework in NVIDIA Isaac Sim [59], demonstrating that the approach can be integrated into a realistic simulation environment.

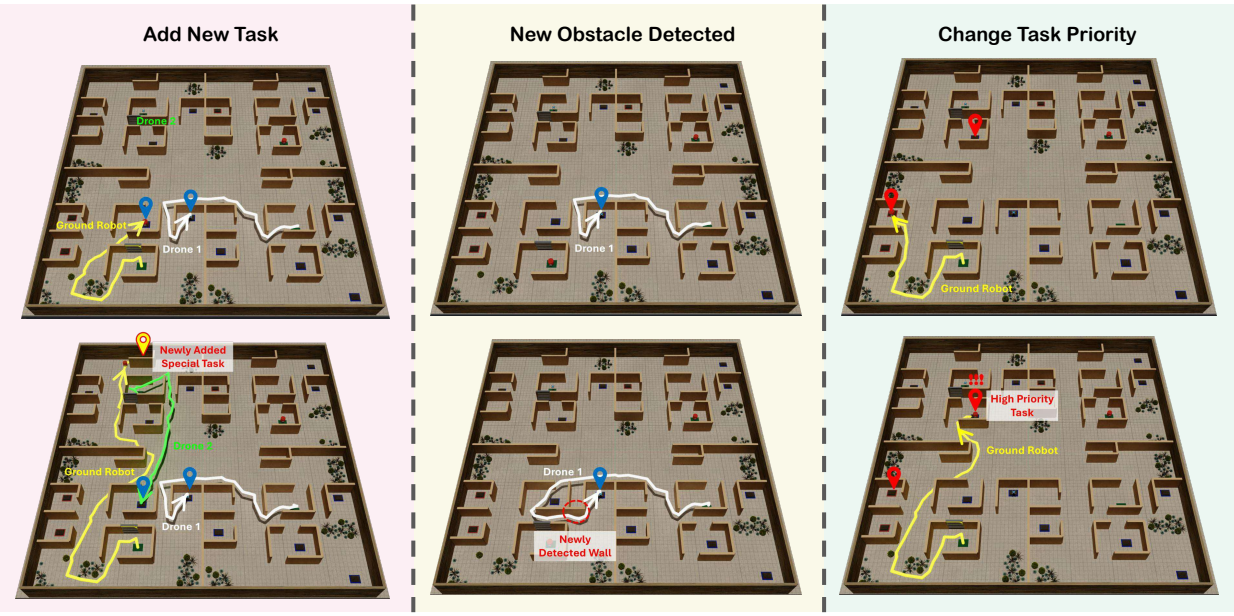



Fig. 9. Demonstration of obstacle-aware task assignment and route replanning under LLM-parsed human instructions in Isaac Sim. The first row shows the initially planned routes, while the second row shows the updated trajectories after executing new instructions: (left) adding a new task, (middle) detecting a new obstacle, and (right) changing task priority. The adaptive planner dynamically adjusts robot paths and task assignments in response to each instruction.

TABLE I
COMPARISON OF METHODS EVALUATED. ALL VARIANTS USE THE SAME PATH PLANNING ALGORITHM.

| Method | Clustering | Intra-Cluster |
|---|---|---|
| CBBA | Bundle | Bundle Order |
| KAN | $k$-means | NN |
| KAM | $k$-means | MILP |
| OATH (ours) | Obstacles-Aware Cluster | MILP |

### A. Methods Compared

1) **CBBA**: the consensus-based bundle algorithm for task assignment.
2) **K-means+Auction+NN (KAN)**: $k$-means clustering, cluster-level auction, and a nearest-neighbor (NN) heuristic for intra-cluster task ordering with Euclidean distances.
3) **K-means+Auction+MILP (KAM)**: $k$-means clustering, cluster-level auction, with an MILP-based intra-cluster solver with Dijkstra distances.
4) **OATH (ours)**: obstacle-aware clustering on the adaptive Halton map, cluster-weighted auction (CWA), and MILP-based intra-cluster selection.

### B. Metrics

We evaluate performance using three metrics and multiple independent randomized task placement trials with identical random seeds across all methods.

1) **Task Assignment Time** ($T_{\text{alloc}}$): wall–clock time spent by the allocator *before* execution begins. This metric focuses purely on the computational efficiency of the allocation algorithm and indicates how quickly each method can produce an allocation in real time, independent of the resulting solution quality.
2) **Total Running Steps** ($S_{\text{total}}$): the sum of discrete execution steps taken by all robots until all tasks are completed,

$$S_{\text{total}} = \sum_{i=1}^{N} s_i,$$

where $s_r$ is the number of steps executed by robot $r$. Since all methods use the same low-level planner (LTL-D* [58]), $S_{\text{total}}$ primarily reflects the efficiency of task assignment and intra-cluster selection, where more optimal allocators enable the team to complete all tasks with fewer overall movements.
3) **Total Running Time** ($T_{\text{total}}$): end-to-end wall–clock mission time from the start of allocation to the completion of the last task,

$$T_{\text{total}} = T_{\text{alloc}} + T_{\text{plan+exec}},$$

where $T_{\text{plan+exec}}$ includes the time of LTL-D* planning/replanning and simulated execution. This metric evaluates not only the optimality of the task assignment results but also accounts for the time spent on task assignment, automaton construction, and replanning, and therefore it reflects the overall efficiency of the proposed algorithm.

### C. Experimental Design

We organize our evaluation into four groups of studies, targeting complementary aspects of the system: allocation efficiency, end-to-end performance, scalability, and LLM-guided instruction handling.

1) **Task Assignment Algorithm Efficiency.** We conduct three controlled experiments and test $T_{\text{alloc}}$ for each algorithm:

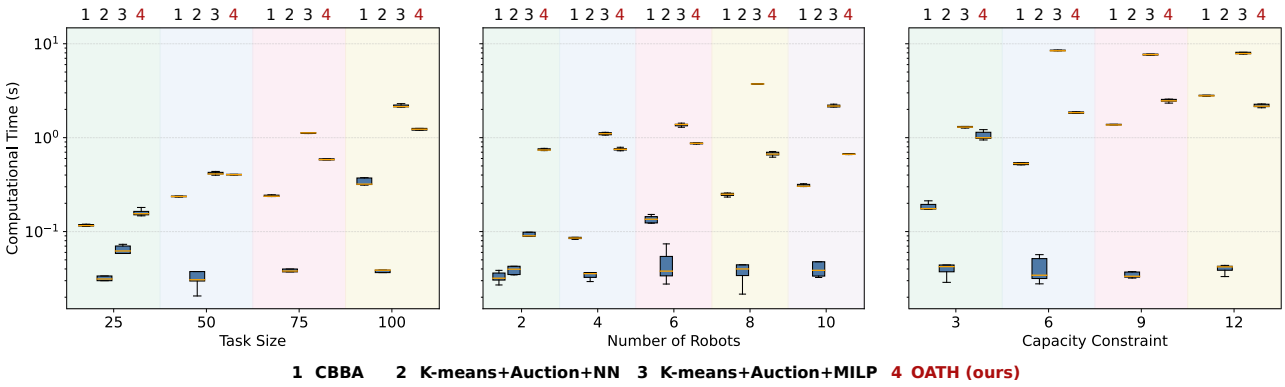



Fig. 10. Computational time of the task assignment algorithm under different experimental settings. Left: varying the task size while keeping the number of robots fixed at 10 and the capacity fixed at 5. Middle: varying the number of robots while keeping the task size fixed at 100 and the capacity fixed at 5. Right: varying the capacity while keeping the task size fixed at 120 and the number of robots fixed at 10. All results are shown as boxplots to illustrate the distribution of running times across different scenarios.

a) *Varying Task Number:* fix the number of robots $R$ and the per-robot capacity $Q$, and increase the number of tasks $P$ to examine sensitivity to problem size.
b) *Varying Robot Number:* fix $P$ and $Q$, and increase the number of robots $R$ to examine sensitivity to team size.
c) *Varying Capacity:* fix $R$ and $P$, and vary the per-robot capacity $Q$. Larger $Q$ increases the complexity of assignment, directly testing allocator robustness.

2) **End-to-End Performance.**
To evaluate the end-to-end performance of our framework, we conduct two sets of experiments.
a) We assess the overall system performance, including *map construction, task assignment, path planning, and execution*.
b) We assess the solution quality of OATH on small-scale problem instances by comparing the total execution distance against an optimization-based MILP baseline, which is solved directly using a commercial solver.

3) **Scalability of OATH.** We evaluate the scalability of the proposed method (OATH) under large task regimes through two experiments.
a) We examine the MILP solver time with respect to robot task capacity.
b) We increase the number of tasks $P$ from 25 to 100 on the Halton map and measure both the total number of steps $S_{\text{total}}$ and the total completion time $T_{\text{total}}$.

4) **LLM-Guided Instruction Performance.** We evaluate the performance of the LLM-guided instruction module through two experiments. The first experiment measures the intent parsing accuracy of the LLM parser, and the second examines the impact of LLM-guided human instructions on overall system behavior.
(i) To evaluate intent parsing accuracy, we construct a test set of 90 natural-language instructions, consisting of three intent categories: *add new task*, *new obstacle detected*, and *change task priority*, with 30 instructions per category. For each intent, the instructions include both clearly specified commands and intentionally ambiguous, colloquial commands to reflect realistic human inputs. Examples include a clear instruction such as "Create a new task at $(5, 16)$ and send it to room B," and a more ambiguous one such as "A new job popped up close to room C." Similar clear and ambiguous instructions are designed for obstacle updates and priority changes. All instructions are processed by the LLM parser, and we record the predicted intent for each input.
(ii) We further investigate the impact of LLM-guided human instructions on system behavior. We consider three representative instruction types: *Add New Tasks*, *New Obstacles Detected*, and *Change Task Priority*. For each type, we compare two modes: a *system built-in baseline* and an *LLM-guided setting*. All experiments are conducted with $R = 4$ robots and $P = 25$ tasks. Each condition is repeated five times, with human instructions intervened at varying time points. We report both $S_{\text{total}}$ and $T_{\text{total}}$ as evaluation metrics to capture efficiency and solution quality. The designs are as follows:
a) *Add New Tasks.* In the baseline, the full set of 25 tasks is available from the start. In the LLM-guided setting, only 24 tasks are initialized, and one additional task is introduced during execution via human instruction.
b) *New Obstacles Detected.* In the baseline, the map is initialized with the obstacles already present. In the LLM-guided setting, the obstacles are initially absent and introduced dynamically during execution.
c) *Change Task Priority.* In the baseline, task priorities remain fixed. In the LLM-guided setting, a human instruction dynamically modifies the priority of one task during execution.

In addition to evaluating intent parsing accuracy and system-level performance impact, we undertake a user study to quantify human-in-the-loop efficiency and analyze the latency characteristics of the LLM-guided pipeline. The detailed experimental setup and corresponding results are provided in Appendix C.

## VII. Results

### A. Parameters Used in Experiments

As shown in Fig. 3, we summarize in Table II the parameter settings used throughout all experiments. These values are fixed unless otherwise specified.

TABLE II
PARAMETERS USED IN EXPERIMENTS

| Parameter | Symbol | Value |
|---|---|---|
| Minimum safe distance | $\delta_{\min}$ | 0.3 |
| Optimal sampling distance | $\delta_{\text{opt}}$ | 0.4 |
| Gaussian width | $\sigma$ | 0.5 |
| Baseline acceptance probability | $\beta$ | 0.2 |
| Number of Halton points | – | 2000 / 1000 |
| Type density weight | $\theta$ | 0.1 |

*Note:* Halton points = 2000 for task assignment and scalability experiments; 1000 for end-to-end and LLM experiments.

### B. Results on Task Assignment Algorithm Efficiency

In the first group of experiments, we evaluate the computational efficiency of the task assignment algorithms under increasing complexity. Three separate studies are conducted by varying one of the following parameters while keeping the others fixed: $P$, $R$, and $Q$. The results are shown in Fig. 10.

Although OATH does not achieve the lowest absolute runtime in simple cases, its runtime growth is much flatter compared with CBBA and KAM. To quantify this observation, we define a *complexity sensitivity index* as

$$\text{Sensitivity} = \frac{T_{\max} - T_{\min}}{\Delta x}, \tag{11}$$

where $T_{\max}$ and $T_{\min}$ are the runtimes under the most complex and simplest configurations, respectively, and $\Delta x$ denotes the corresponding change in the complexity parameter (e.g., $P$, $R$, or $Q$). A lower value indicates that the method's runtime grows more gently as the problem complexity increases.

TABLE III
COMPLEXITY SENSITIVITY OF DIFFERENT ALLOCATORS, DEFINED AS $(T_{\max} - T_{\min})/\Delta x$ UNDER VARYING TASK NUMBER, ROBOT NUMBER, AND CAPACITY. LOWER VALUES INDICATE BETTER ROBUSTNESS TO COMPLEXITY GROWTH.

| Method | Task Number ($P$) | Robot Number ($R$) | Capacity ($Q$) |
|---|---|---|---|
| CBBA | **0.003** | 0.035 | 0.296 |
| KAN | **5.04E-05** | **0.001** | **0.001** |
| KAM | 0.028 | 0.261 | 0.797 |
| OATH | 0.015 | **0.010** | **0.13** |

Table III summarizes the sensitivity scores. KAN attains the smallest sensitivity overall, but it is a very simple pipeline that prioritizes speed over solution quality: the $k$-means pre-clustering and nearest-neighbor assignment do not optimize an objective, provide no optimality guarantees, and are prone to cluster-imbalance effects; moreover, the consistency of distance matrices used in task assignment and path planning is largely ignored. OATH is not always among the two smallest entries in all matrices; however, its scores remain consistently low — the maximum is only 0.015 — so it is the second smallest overall. Compared with CBBA and KAM, OATH is typically about an order of magnitude less sensitive in most cases, indicating that its allocation module scales more gracefully with problem growth.

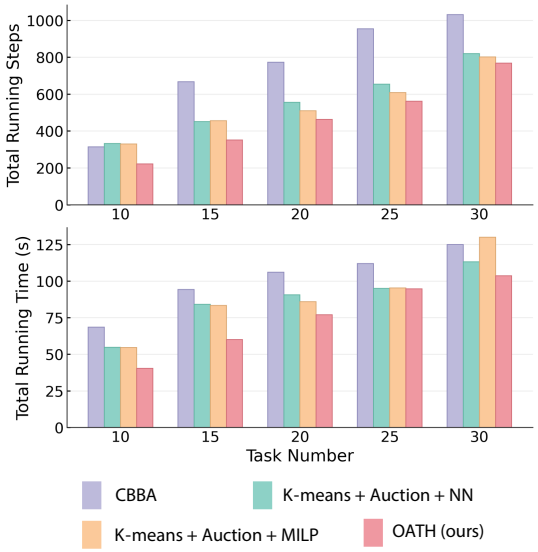



Fig. 11. Experimental results of the task assignment algorithm in a simulated environment. Top: total running steps of the four robots as a function of task number. Bottom: total running time of the four robots as a function of task number.

TABLE IV
TOTAL STEPS AND TOTAL RUNNING TIME (S) VS. TASK NUMBER.

| **Total Steps** | | | | |
|---|---|---|---|---|
| Tasks | CBBA | KAN | KAM | OATH |
| 10 | 314 | 332 | 330 | **222** |
| 15 | 667 | 451 | 456 | **351** |
| 20 | 773 | 555 | 510 | **463** |
| 25 | 954 | 654 | 608 | **562** |
| 30 | 1031 | 820 | 802 | **768** |
| **Total Running Time (s)** | | | | |
| Tasks | CBBA | KAN | KAM | OATH |
| 10 | 68.53 | 54.79 | 54.60 | **40.36** |
| 15 | 94.20 | 84.13 | 83.30 | **60.11** |
| 20 | 106.02 | 90.66 | 85.96 | **76.98** |
| 25 | 111.96 | 95.02 | 95.35 | **94.76** |
| 30 | 124.94 | 113.14 | 129.95 | **103.63** |

### C. Results on End-to-End Computational Performance

The second group of experiments evaluates the overall planner performance, including map construction, task assignment, planning, and execution. Two metrics are considered: $T_{\text{total}}$ and $S_{\text{total}}$.

As shown in Table IV and Fig 11, OATH consistently achieves the lowest number of total running steps, indicating that it generates the most efficient allocations and task sequences. More importantly, OATH also achieves the shortest end-to-end execution time across all task sizes. Compared with CBBA, OATH reduces the mission completion time by an

TABLE V
COMPARISON BETWEEN OATH AND THE MILP BASELINE SOLVED USING GUROBI IN TERMS OF TOTAL EXECUTION DISTANCE AND COMPUTATION TIME.

| Number of Tasks | Total Distance | | | | Time (s) | |
|---|---|---|---|---|---|---|
| | OATH | MILP | Gap to MILP (%) | Gap to LB (%) | OATH | MILP |
| 9 | 88.108 | 67.043 | 23.91 | 0.00 | 0.0716 | 3.96 |
| 12 | 102.446 | 85.381 | 16.66 | 0.00 | 0.0837 | 32.94 |
| 15 | 125.051 | 108.226 | 13.45 | 6.07 | 0.2011 | $\geq$600 |
| 18 | 146.529 | 133.006 | 9.23 | 16.69 | 0.3237 | $\geq$600 |
| 21 | 208.357 | 155.572 | 25.33 | 22.12 | 0.3663 | $\geq$600 |

average of **27.4%**, even though its allocation phase consumes slightly more computation time.

These results highlight that the overall mission efficiency depends not only on the speed of the allocation process but also, on how well the allocation accounts for the real execution paths. In practice, it is worthwhile to spend additional time in task assignment to obtain higher allocation optimality, since the resulting reduction in execution time outweighs the extra allocation overhead. This confirms that OATH provides the best trade-off between allocation cost and overall mission efficiency.

To further evaluate the solution quality of OATH, we added a small-scale comparison against an optimization-based MILP baseline solved using Gurobi. In this experiment, we compare OATH with the MILP solution in terms of total travel distance and computation time for problem instances with $9-21$ tasks. The results are summarized in Table V. For small instances (9 and 12 tasks), where the MILP solver terminates with certified optimality, OATH produces feasible solutions with relative distance gaps of $23.91\%$ and $16.66\%$, respectively. While OATH does not achieve global optimality in these cases, it computes solutions within $0.07-0.08$ seconds, compared to $3.96-32.94$ seconds for MILP. This highlights a clear trade-off: OATH sacrifices a portion of optimality in exchange for significantly faster computation.

For larger instances (15 tasks and above), the MILP solver reaches the imposed 600-second time limit and does not return a certified optimal solution within the allowed runtime. To provide context, we report the remaining optimality gap of the MILP solution with respect to the best-known lower bound, which ranges from $6.07\%$ to $22.12\%$. This indicates that even the optimization-based baseline becomes computationally impractical in this regime. In contrast, OATH consistently produces feasible solutions within sub-second computation time ($0.20-0.37$ seconds), while maintaining competitive solution quality relative to the best MILP solutions obtained within the time limit. These results show that OATH provides a good trade-off between solution quality and computation time, which is essential for real-time heterogeneous task allocation.

### D. Results on Scalability Performance

To evaluate the scalability of OATH, we conduct two experiments. The first experiment examines the MILP solver time, while the second evaluates the overall system performance as the number of tasks increases.

The MILP solver time is an important factor affecting the scalability of OATH. In the end-to-end experiments, the cluster size does not exceed 10 tasks per cluster. To further examine the practical scaling behavior of the MILP solver, we additionally evaluate scenarios where a single cluster contains up to 18 tasks. In this experiment, we gradually increase the number of tasks that the MILP needs to sequence (corresponding to the robot task capacity), and report the solver time over five trials for each setting. The results are shown in Fig.12.

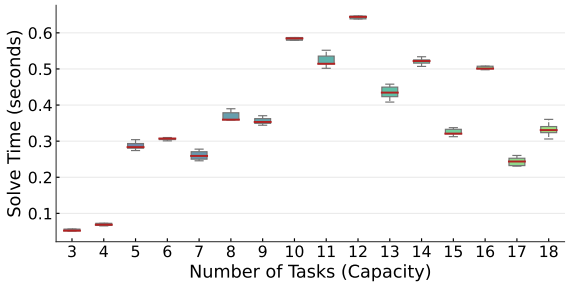



Fig. 12. MILP solver runtime under varying task capacities. Boxplots illustrate the distribution of solver runtimes over five independent trials for each task capacity, with the red line indicating the median solve time.

As illustrated in Fig. 12, the MILP solver time initially increases as the number of tasks grows, since the problem size and the number of feasible task sequences increase. The solver time reaches its maximum when the number of tasks is around 12, and remains below $0.7$ seconds in all tested cases. Interestingly, when the number of tasks further increases, the solver time decreases. This is because when the number of tasks to be sequenced approaches the total number of tasks in the cluster, the combinatorial flexibility of the MILP is reduced: most tasks must be included in the route to satisfy the capacity constraint, leaving fewer alternative task subsets and permutations to consider. As a result, the MILP search space becomes more constrained, and the solver effectively focuses on excluding a small number of tasks rather than exploring many competing task sequences, which leads to faster convergence.

We also assess the scalability of OATH by increasing the number of tasks from $P$=25 to $P$=100 while keeping the team configuration and planner fixed. As shown in Fig. 13, both the $T_{\text{total}}$ and $S_{\text{total}}$ grow approximately linearly with $P$, indicating acceptable scaling behavior for online use. Concretely, the total time increases from 109 s to 255 s ($\times 2.34$) and the total steps from 917 to 2543 ($\times 2.77$) as $T$ quadruples ($\times 4$). The amortized cost per task improves with scale: steps per task decrease from 36.7 to 25.4 ($-30.7\%$), and time per task

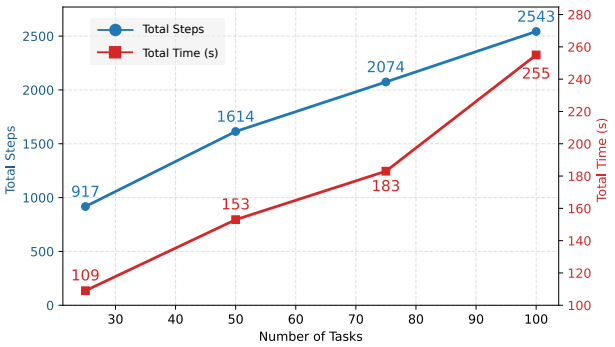



Fig. 13. Scalability of OATH with respect to the number of tasks. With the team and planner fixed, increasing $P$ from 25 to 100 yields approximately linear growth in *total steps* (left axis) and *total time* (right axis). The near-linear trend and improving per-task cost indicate that OATH scales well to larger workloads.

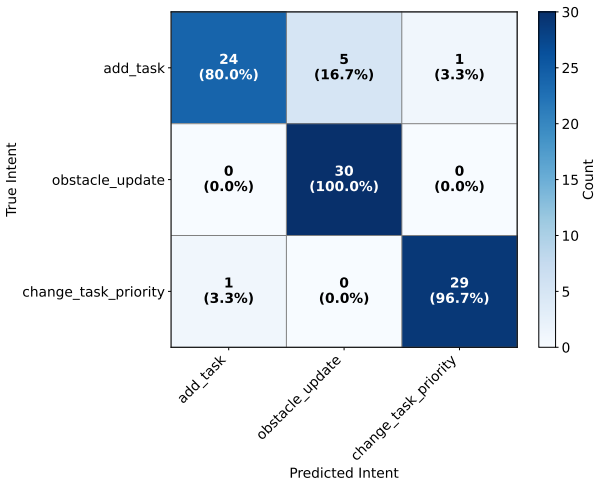



Fig. 14. Confusion matrix for LLM intent parsing. The matrix reports predicted versus ground-truth instruction intents over 90 test commands, including add new task, obstacle update, and change task priority. Misclassifications occur only for highly ambiguous inputs, while obstacle update instructions are parsed with 100% accuracy.

drops from $4.36\,\mathrm{s}$ to $2.55\,\mathrm{s}$ ($-41.5\%$). These trends indicate that OATH exhibits near-linear growth in overall effort while the time and steps per task decrease, demonstrating strong scalability.

### *E. Results on LLM-Guided Instruction Performance*

(i) To evaluate the accuracy of LLM intent parsing, we summarize the parsing results using a confusion matrix, shown in Fig. 14. The parser achieves $100\%$ accuracy for the obstacle update intent. Misclassifications only occur between the *add new task* and *change task priority* intents. Specifically, among the *add new task* instructions, five are misclassified as obstacle updates and one is misclassified as a change task priority. For the *change task priority* intent, one instruction is misinterpreted as adding a new task. All misclassified cases correspond to highly ambiguous natural-language inputs. For example, the instruction "There is a new situation near $(7, 15)$ that needs attention" is interpreted as an obstacle update because the phrase "new situation," together with the absence of task-specific keywords, suggests a change in the environment rather than the creation of a new task. Similarly, an instruction such as "Something else probably needs to be handled now," which is labeled as a change task priority, does not specify whether an existing task should be reprioritized or whether a new task should be created. Due to the lack of explicit semantic cues, the parser interprets this input as adding a new task.

Overall, these results show that the LLM parser performs reliably across a wide range of natural-language instructions. Parsing errors arise primarily when the input is underspecified and lacks clear intent-related cues. In such cases, the system does not execute the parsed command directly. Instead, it triggers an interactive clarification step to resolve the ambiguity (see Appendix B). This reduces the risk of incorrect execution and improves system reliability.

(ii) We evaluate LLM-guided instruction performance using two metrics $T_{\text{total}}$ and $S_{\text{total}}$, as shown in Fig. 15. Overall, the LLM-guided settings show larger variance than the system built-in baselines across all instruction types. This variability mainly stems from the timing and context of human interventions, which influence how the system parses instructions and updates the map, task assignment, or execution plan. Specifically:

a) *Add New Tasks:* The system built-in baseline completes 25 tasks efficiently, while the LLM-guided setting (initially 24 tasks) incurs overhead when the new task is introduced online. The insertion timing strongly influences results: if the new task is added when robots are already near the task region, the additional cost is small; otherwise, it can cause longer detours and higher variance in running time.

b) *New Obstacles Detected:* The system built-in baseline achieves stable performance since obstacles are encoded from the beginning. In contrast, the LLM-guided setting shows greater variance in both time and steps. Time variance arises because, when new obstacles appear, robots must pause, check route intersections, and rebuild the product automaton with the updated map. Step variance (the total steps can be smaller than the baseline, as shown in Fig. 15) occurs because if obstacles emerge later, many tasks may already be completed through shortcuts in those regions.

c) *Change Task Priority:* The system built-in baseline benefits from a fixed, globally optimized priority order. In contrast, the LLM-guided setting must dynamically reorder tasks at execution time. This can force robots to abandon ongoing trajectories and replan, leading to significantly larger variance in both total time and steps, and in some cases noticeably higher overhead than the baseline.

Overall, these results indicate that while the LLM-guided system enables flexible online adaptation to human instructions, integrating human interaction into MATP introduces a trade-off between adaptability and optimality, as it adds overhead time and increases planning uncertainty.

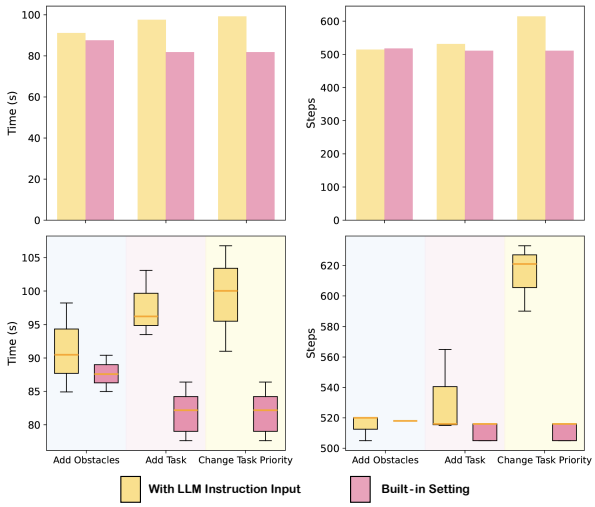



Fig. 15. Comparison between *human-issued* and *system built-in* instructions for three instruction types: *New Obstacles Detected*, *Add New Tasks*, and *Change Task Priority*. The left column reports execution time (s), and the right column reports the total number of robot steps. The top row shows means (bar plots), and the bottom row shows variability across trials (box plots). Yellow and pink bars/boxes denote the two modes, respectively, as indicated in the legend.

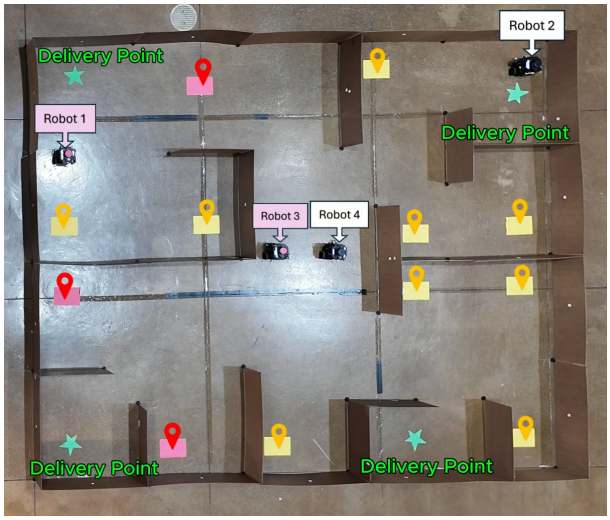



Fig. 16. Hardware experimental setup with four TurtleBot3 Burger robots deployed in a $3.66 \times 4.57$ m indoor environment. Brown panels indicate static obstacles that form corridor structures. Yellow markers represent regular pickup tasks, while pink markers indicate special tasks. Four delivery locations are denoted by green stars. Robots with pink identifiers are capable of executing both task types, whereas the remaining robots are restricted to yellow tasks only. All task allocation and planning are executed online, and robots receive goal positions sequentially from the planner during operation.

## VIII. Hardware Validation

To further evaluate the practical applicability of the proposed framework, we carry out real-world experiments using four TurtleBot3 Burger platforms. All experiments are performed fully online, including task allocation, path planning, and LLM-guided instruction processing. No task assignments or trajectories are precomputed offline.

### A. Experimental Setup

The experiments run in an indoor workspace measuring approximately $3.66 \times 4.57$ m ($12 \times 15$ ft). The environment contains two types of pickup tasks distinguished by color and four delivery locations. In the physical setup, regular tasks are marked in yellow, as shown in Fig. 16. Four TurtleBot3 Burger robots are deployed with heterogeneous capabilities. Two robots (identified with pink markers) can execute both task types, while the remaining two robots can only execute the yellow task type. This configuration directly corresponds to the heterogeneous capability matrix used in the planner.

All components of OATH operate online within a hybrid centralized–distributed architecture. Obstacle-aware clustering and the cluster-weighted auction (CWA) are executed on a central workstation to determine cluster assignments among robots. After cluster assignment, each robot runs an independent ROS node that communicates with the central planner. The intra-cluster MILP task selection and global path planning are computed by the planner, while each robot maintains its own execution node responsible for receiving goals, monitoring progress, and reporting task completion. The planner dispatches goal positions sequentially to each robot according to its assigned task sequence. Each TurtleBot navigates to the received goal using ROS2 Navigation2. Upon reaching the assigned goal, the robot sends a `goal_reached` signal back to the planner, which updates the task state and issues the next target if needed. This closed-loop interaction continues until all tasks are completed.

### B. Hardware Experiment 1: Online Task Allocation and Execution

In the first hardware experiment, we evaluate the complete allocation and execution pipeline with four robots and twelve tasks. Each robot possesses a carrying capacity of two tasks. The system successfully performs obstacle-aware clustering, heterogeneous cluster assignment, and intra-cluster sequencing online.

Throughout the experiment, robots strictly respect capability constraints. Robots without pink markers never attempt incompatible tasks, while pink-marked robots handle both types when necessary. The planner–robot interaction remains stable during iterative goal dispatch and feedback, and no task synchronization issues are observed. The experiment video is shown in the project website https://llm-oath.github.io/.

### C. Hardware Experiment 2: LLM-Guided Replanning

Next, in the second experiment, we validate the LLM-guided planner on the hardware, where human instructions issued during execution are parsed into structured intents that trigger online task reassignment or replanning.

In this experiment, two representative instructions are introduced: (1) the detection of a new obstacle and (2) the insertion of a new task. Upon receiving each instruction, the planner updates the task allocation and path plan accordingly in real time. The updated goal positions are dispatched to the corresponding robots without interrupting ongoing execution. After receiving new goals, the robots smoothly switch to

their updated target positions using ROS 2 Navigation2. The hardware execution remains stable throughout the replanning process, indicating that the LLM-based instruction module integrates consistently with the real robotic execution stack. The experiment video is available on the project website at https://llm-oath.github.io/.

### D. Robustness to Communication and Hardware Constraints

Unlike simulation environments that assume ideal communication, real robot systems introduce network latency, packet variability, and sensing noise. The hardware experiments allow us to examine how these practical factors influence system behavior beyond simulation assumptions.

First, the proposed framework does not rely on high-frequency distributed consensus. The cluster-weighted auction is executed centrally and only when a new allocation cycle is triggered. Communication between the planner and robots is limited to goal dispatch and goal-reached signals, which are low-bandwidth and event-driven. In our four-robot experiments, this communication pattern did not lead to observable desynchronization. However, as the number of robots increases, the central workstation must handle a larger number of goal dispatch messages, completion signals, and state updates. In wireless networks, higher traffic density may introduce packet delay or temporary congestion, increasing the time between task completion and subsequent allocation updates. While the event-driven architecture avoids iterative consensus overhead, increased communication delay could slow down reallocation cycles when scaling to many robots. In such scenarios, distributed clustering or localized auction mechanisms could be considered to reduce centralized communication load and maintain scalability.

Second, computational complexity remains moderate due to the hierarchical design. Clustering, scoring, and intra-cluster MILP operate on bounded task subsets rather than the full task set. In our hardware experiments, planning steps consistently finish before robots reach their next goals. Nevertheless, in larger environments with denser task distributions or increased robot counts, computation time may grow, potentially increasing the interval between allocation cycles. The current architecture remains practical at the tested scale, but further decentralization may be beneficial for large-scale deployments.

Third, sensing noise and kinematic deviations are handled at the control layer by the ROS2 Navigation2 stack, while OATH operates at the task-allocation and waypoint level. Moderate localization noise does not invalidate task feasibility, since allocation decisions rely on estimated path costs rather than instantaneous pose measurements. However, larger localization drift or traversal-time deviations may reduce allocation optimality by altering effective travel costs. Additionally, heterogeneous execution speeds may introduce variability in task completion order. The event-driven reallocation mechanism accommodates such asynchrony naturally: allocation cycles are triggered by task completion rather than synchronized clocks. This prevents global deadlock, although efficiency may decrease if execution variance becomes significant.

Finally, compared with simulation, physical robots operate at lower speeds and under real sensing uncertainty. While this often provides additional planning margins, it also introduces variability in traversal time. The hardware experiments suggest that the framework remains functional under moderate real-world uncertainty. At larger scales or under harsher communication and sensing conditions, performance may degrade gradually in terms of allocation optimality or responsiveness, rather than feasibility.

Overall, the hardware experiments demonstrate that the OATH framework maintains consistent allocation and replanning behavior under realistic communication conditions and sensing uncertainty at the tested scale. These results provide empirical evidence toward bridging the gap between simulation and real-world deployment, while also highlighting considerations for larger-scale implementations.

## IX. Conclusion

This work presents OATH, an obstacle-aware task assignment and planning framework for MATP in heterogeneous robot teams. It introduces an adaptive Halton sequence map that adjusts sampling density according to obstacle distribution and a hierarchical cluster–auction–task selection scheme that generalizes to arbitrary task types while reducing allocation complexity. Together, these components enable scalable and suboptimal task assignment.

Beyond task assignment, OATH integrates LLMs as persistent interpreters during execution. Unlike prior approaches that use LLMs only for initialization, OATH continuously translates natural language instructions into structured constraints and task updates, enabling adaptation to changing human intent, unforeseen obstacles, and mission updates.

Extensive simulations in Isaac Lab demonstrate that OATH achieves superior performance over baseline methods in scalability, and responsiveness to environmental changes, successfully managing up to 100 tasks with real-time performance. Beyond simulation, we validate OATH through online hardware experiments. The results demonstrate stable task allocation, online replanning, and LLM-guided intervention under realistic communication latency and execution uncertainty, confirming the framework's reliability in real-world deployment.

In future work, we plan to further investigate learning-based approaches and extend the system to highly heterogeneous teams, including humanoid robots [60], [61].

## Appendix A
## Asymptotic Complexity Analysis

Let $N$ denote the number of roadmap samples, $P = |\mathcal{P}|$ the number of tasks, $R = |\mathcal{R}|$ the number of robots, and $K$ the number of clusters.

### Time Complexity

**Roadmap construction.** Planar Delaunay triangulation requires $T_{\text{tri}} = \mathcal{O}(N \log N)$ [62]. For planar graphs, $E \leq 3N - 6 = \mathcal{O}(N)$ [63].

**Distance matrix construction.** Dijkstra with a binary heap runs in $\mathcal{O}((|V| + |E|) \log |V|)$ [64]. With $|V| = N$ and $E =$

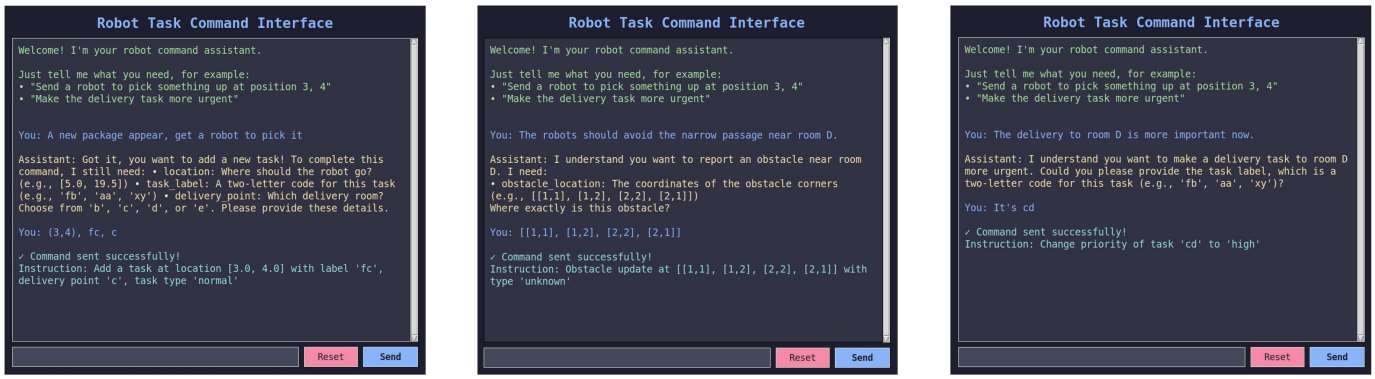



Fig. 17. Interactive human–LLM command interface under three instruction types: (left) adding a new task, (center) updating obstacle information, and (right) changing task priority. The interface exposes the LLM's parsed interpretation for human confirmation before execution.

$\mathcal{O}(N)$, we obtain $T_{\text{ssp}} = \mathcal{O}(N \log N)$. Executed for $P$ sources, $T_{\mathcal{M}} = \mathcal{O}(PN \log N)$, hence $T_{\text{pre}} = \mathcal{O}(PN \log N)$.

**Clustering.** Agglomerative hierarchical clustering requires $T_{\text{cluster}} = \mathcal{O}(P^2 \log P)$ [54], [64].

**Auction.** Scoring and coordination yield $T_{\text{CWA}} = \mathcal{O}(IRK)$.

**Intra-cluster MILP.** For robot $r$ with $n_r$ tasks [65],

$$(2n_r)! \approx \sqrt{4\pi n_r} \left(\frac{2n_r}{e}\right)^{2n_r} \tag{12}$$

Thus $T_{\text{MILP},r} = \mathcal{O}(c^{n_r}),\ c > 1$, consistent with NP-hard routing problems [66]. In OATH, clustering constrains $n_r$ to remain small, so the MILP size is structurally bounded in practice.

**Product automaton.** The state space satisfies $|\mathcal{V}_{prod}| = |\mathcal{V}_B|N$ [67].

Graph search complexity is [64]

$$T_{\text{plan}} = \mathcal{O}((|\mathcal{V}_{prod}| + |\mathcal{E}_{prod}|) \log |\mathcal{V}_{prod}|) \tag{13}$$

*Overall Time Complexity*

Initial allocation cycle:

$$\begin{aligned} T_{\text{init}} = {} & \mathcal{O}(PN \log N) + \mathcal{O}(P^2 \log P) \\ & + \mathcal{O}(IRK) + \sum_{r=1}^{R} \mathcal{O}(c^{n_r}) \\ & + \mathcal{O}((|\mathcal{V}_{prod}| + |\mathcal{E}_{prod}|) \log |\mathcal{V}_{prod}|)\,. \end{aligned} \tag{14}$$

Subsequent cycles:

$$\begin{aligned} T_{\text{cycle}} = {} & \mathcal{O}(P^2 \log P) + \mathcal{O}(IRK) \\ & + \sum_{r=1}^{R} \mathcal{O}(c^{n_r}) \\ & + \mathcal{O}((|\mathcal{V}_{prod}| + |\mathcal{E}_{prod}|) \log |\mathcal{V}_{prod}|)\,. \end{aligned} \tag{15}$$

*Space Complexity*

Roadmap storage requires $\mathcal{O}(N)$. The distance matrix requires $\mathcal{O}(P^2)$. The product automaton requires $\mathcal{O}(|\mathcal{V}_B|N)$ [67]. Total memory:

$$\mathcal{O}(N + P^2 + |\mathcal{V}_B|N) \tag{16}$$

## APPENDIX B
## INTERACTIVE HUMAN–LLM COMMAND INTERFACE AND PROMPT DESIGN

To support more natural and less structured user inputs during task execution, we implement an interactive human–LLM command interface that complements the instruction-parsing pipeline described in Sec. V. Rather than requiring users to provide fully specified commands in a single utterance, the interface accepts high-level, ambiguous, or incomplete instructions and supports a short clarification dialogue between the user and the LLM before committing any update to the planner.

The interaction follows a turn-based design, as illustrated in Fig. 17. When a user issues an instruction, the LLM first infers the underlying intent (for example, adding a task, updating obstacles, or changing task priority) and extracts all parameters that can be reliably determined from the input. If one or more required parameters are missing or ambiguous, such as task location, delivery target, or obstacle position, the LLM explicitly requests additional information from the user. Only after all required fields are resolved does the system generate a structured command in the predefined JSON format and dispatch it to the planner.

In our implementation, the LLM acts strictly as an instruction parser rather than a planner. After receiving a human command, it performs intent detection and slot filling to verify whether all required fields (for example, task location, task label, and delivery point) are specified. If critical information is missing or underspecified, the system does not generate values implicitly. Instead, it asks targeted follow-up questions to resolve the ambiguity before issuing a structured command to the task allocation module. An example interaction is shown in Fig. 17.

For highly ambiguous commands such as "get that robot over there to handle it," the system correctly identifies that essential grounding information is missing, including which robot is being referenced and what location "there" corresponds to. Because the LLM does not have access to a fully grounded world model or real-time robot state information, it cannot resolve such references autonomously. This limitation represents a known failure mode. In these cases, the system relies on user clarification rather than making implicit assump-

tions. We intentionally adopt this conservative design choice to avoid incorrect task execution.

This design provides three practical advantages. First, it lowers the cognitive burden on human operators by removing the need to remember rigid command templates. Second, it improves system robustness by preventing partially specified or ill-formed commands from directly triggering replanning or task reassignment. Third, the interface explicitly exposes the LLM's parsed interpretation of the instruction to the human commander for confirmation, enabling users to verify correctness before execution and reducing the risk of downstream errors caused by misinterpretation.

The LLM is guided by a fixed system prompt that defines its role as an interactive command parser and dialogue manager. The complete prompt used in our experiments is provided below.

*a) User Interface Prompt.:* This prompt is used by the UI to guide the dialogue with the human commander.

```
You are a friendly and intelligent assistant helping
users control robot team. You can understand casual,
vague, or informal language and interpret the user's
intent.
YOUR CAPABILITIES:
You support three types of robot commands:
1. add_task
   REQUIRED:
   - location: [x, y] coordinates.
   - task_label: a two-letter code defined by the user.
   - delivery_point: one of {'b','c','d','e'}
     'b' -> [6, 3]
     'c' -> [5, 16]
     'd' -> [16, 13]
     'e' -> [16, 5.5]
   OPTIONAL:
   - task_type: {'normal','special'} ('normal' by
default)
2. obstacle_update
   REQUIRED:
   - obstacle_location: line segment or polygon.
   OPTIONAL:
   - obstacle_type (e.g., 'wall', 'block', 'unknown')
3. change_task_priority
   REQUIRED:
   - task_label
UNDERSTANDING CASUAL LANGUAGE:
- "go grab something from point A" -> add_task
- "there's something blocking the way near X,Y" ->
obstacle_update
- "make task X more urgent" -> change_task_priority
CONVERSATION STYLE:
- Be conversational and friendly.
- If the user is vague, make reasonable guesses and
confirm.
- Use context from previous turns.
WHEN INFORMATION IS MISSING:
If the intent is clear but required parameters are
missing:
1. State what you understood (intent + known
parameters)
2. List all missing required parameters
3. Provide example formats for each missing parameter
RESPONSE FORMAT (JSON ONLY):
{
  "status": "complete" | "need_more_info" |
"clarification",
  "intent": "add_task" | "obstacle_update" |
"change_task_priority" | null,
  "collected_params": {...},
  "missing_params": [...],
  "message": "response to the user",
  "instruction": "final formatted instruction (only if
complete)"
}
When status is "complete":
- add_task:
  "Add a task at location [x,y] with label 'id',
   delivery point 'd', task type 'type'."
- obstacle_update:
  "Obstacle update at [[x1,y1],[x2,y2],...] with type
'type'."
- change_task_priority:
  "Change priority of task 'id' to high."
CRITICAL:
Respond with ONLY a valid JSON object.
Do not include explanations or formatting outside
JSON.
```

*b) Instruction Parser Prompt.:* After the UI receives a complete instruction from the human commander, the instruction parser reformulates the instruction and forwards it to the planner.

```
You are a command parser. Convert the following
instruction into a JSON array, where each element
represents one intent.
Possible intents: "add_task" or "obstacle_update" or
"change_task_priority". Each element should follow
one of the following formats:
For "add_task":
{{
  "intent": "add_task",
  "parameters": {{
    "location": [1.0, 2.0],
    "delivery_point": "b",
    "task_label": "fb",
    "task_type": "special"
  }}
}}
For "obstacle_update":
{{
  "intent": "obstacle_update",
  "parameters": {{
    "obstacle_location": [[4.1, 1.1], [4.1, 2.0],
[2.5, 2.0], [2.5, 1.1]],
    "obstacle_type": "wall"
  }}
}}
For "change_task_priority":
{{
  "intent": "change_task_priority",
  "parameters": {{
    "task_label": "fb",
    "priority": "high"
  }}
}}
Instruction: {instruction_text}
Return ONLY the JSON array. Do NOT add any
explanation.
```

## APPENDIX C
## LLM-GUIDED INSTRUCTION USER STUDY

In addition to evaluating intent parsing accuracy and system performance impact, we conduct a user study to quantify human-in-the-loop efficiency and analyze the latency of the LLM-guided pipeline.

We recruit eight participants to evaluate the efficiency of human interventions. Each participant performs three trials for each intervention intention, resulting in 24 trials per intention. Participants are allowed to freely design their own instructions in each trial. In particular, they may introduce new tasks or new obstacles at arbitrary locations in the environment, without being restricted to predefined templates or fixed coordinates.

To evaluate the efficiency and latency characteristics of the LLM-guided pipeline, we conduct two complementary experiments. The first experiment examines the overall human-in-the-loop interaction by measuring human formulation time and total system reaction time. The second experiment further

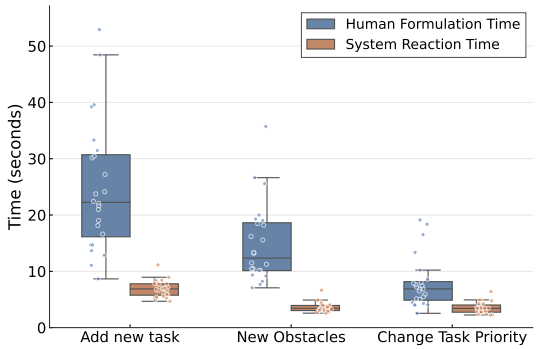



Fig. 18. Comparison between human formulation time and system reaction time for different types of human-in-the-loop interventions. The system reaction time includes both LLM parsing time and the subsequent replanning time. Results are reported for three representative scenarios: adding a new task, introducing new obstacles, and changing task priorities.

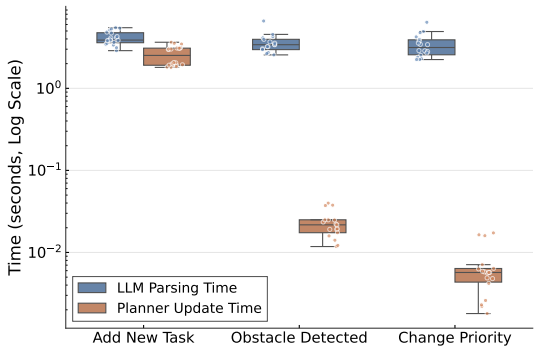



Fig. 19. Breakdown of response latency in LLM-triggered interventions. Blue boxplots denote the LLM inference (natural language parsing) time, while orange boxplots denote the planner update time. The y-axis is shown in logarithmic scale. Across all three intervention types (Add New Task, Obstacle Detected, and Change Priority), the LLM parsing stage dominates the total latency, whereas the replanning module operates at the millisecond level. This separation clarifies that the primary bottleneck arises from model inference rather than from the task allocation algorithms.

decomposes the system reaction time into LLM inference time and planner update time, allowing us to analyze the source of latency under different intervention types.

*a) Human Formulation vs. System Reaction Time.:* The comparison between human formulation time and system reaction time is shown in Fig. 18. Across all three intervention types (*Add New Task*, *New Obstacle Detected*, and *Change Task Priority*), the human formulation time exhibits larger variance than the system reaction time. This variance is expected, as different participants compose instructions at different speeds and with varying levels of detail.

Among the three intervention types, *Add New Task* results in the longest human formulation time. This is consistent with the higher information requirement of this intention, which typically involves specifying task location, delivery point, and task type. In contrast, *New Obstacle Detected* and *Change Task Priority* require fewer parameters and therefore yield shorter formulation times.

The system reaction time remains relatively stable across participants and intervention types. In all cases, the reaction time is consistently lower and less variable than the human formulation time. This indicates that once a valid instruction is provided, the LLM parsing and replanning pipeline introduces limited additional delay.

*b) Latency Decomposition.:* The breakdown of response latency is shown in Fig. 19. The dominant contributor to the total response latency is the LLM inference stage rather than the planner update stage.

More specifically, for the *Add New Task* scenario, both the LLM parsing time and the system response (planner update) are on the order of $10^0$ seconds. The LLM parsing typically falls in the several-second range (roughly 2–4 s), while the system response time is also seconds-level (roughly 1–3 s). This indicates that adding a new task triggers a substantially heavier planning workload, including inserting a new node into the Halton-based roadmap, updating the task-to-task distance structure, performing task reallocation, and recomputing the corresponding routes, so the end-to-end latency is not dominated by LLM inference alone in this case.

In contrast, for the *Obstacle Detected* and *Change Priority* scenario, the system response time drops to approximately $10^{-3}$–$10^{-2}$ seconds, which is about two orders of magnitude smaller than the LLM parsing time.

Overall, these results indicate that the primary latency source in the human-in-the-loop setting is the natural language parsing stage rather than the task allocation or replanning algorithms. The planner update remains computationally lightweight relative to the LLM inference overhead.

## REFERENCES

[1] H. Chakraa, F. Guérin, E. Leclercq, and D. Lefebvre, "Optimization techniques for multi-robot task allocation problems: Review on the state-of-the-art," *Robotics and Autonomous Systems*, vol. 168, p. 104492, 2023.

[2] C. Ju, J. Kim, J. Seol, and H. I. Son, "A review on multirobot systems in agriculture," *Computers and Electronics in Agriculture*, vol. 202, p. 107336, 2022.

[3] G. P. Das, T. M. McGinnity, S. A. Coleman, and L. Behera, "A distributed task allocation algorithm for a multi-robot system in healthcare facilities," *Journal of Intelligent & Robotic Systems*, vol. 80, pp. 33–58, 2015.

[4] B. Mishra, D. Garg, P. Narang, and V. Mishra, "Drone-surveillance for search and rescue in natural disaster," *Computer Communications*, vol. 156, pp. 1–10, 2020.

[5] M. E. Cao, X. Ni, J. Warnke, Y. Han, S. Coogan, and Y. Zhao, "Leveraging heterogeneous capabilities in multi-agent systems for environmental conflict resolution," in *2022 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2022, pp. 94–101.

[6] O. Salzman and R. Stern, "Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020, pp. 1711–1715.

[7] A. Shamsah, J. Jiang, Z. Yoon, S. Coogan, and Y. Zhao, "Terrain-aware model predictive control of heterogeneous bipedal and aerial robot coordination for search and rescue tasks," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 12 352–12 358.

[8] Z. Zhou, D. J. Lee, Y. Yoshinaga, S. Balakirsky, D. Guo, and Y. Zhao, "Reactive task allocation and planning for quadrupedal and wheeled robot teaming," in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2022, pp. 2110–2117.

[9] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, and P. J. Stuckey, "Integrated task assignment and path planning for capacitated multi-agent pickup and delivery," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5816–5823, 2021.

[10] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE transactions on robotics*, vol. 25, no. 4, pp. 912–926, 2009.

[11] M. De Ryck, D. Pissoort, T. Holvoet, and E. Demeester, "Decentral task allocation for industrial agv-systems with routing constraints," *Journal of Manufacturing Systems*, vol. 62, pp. 135–144, 2022.

[12] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithms for multirobot task assignment with task deadline constraints," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 876–888, 2015.

[13] K. Padmanabhan Panchu, M. Rajmohan, R. Sundar, and R. Baskaran, "Multi-objective optimisation of multi-robot task allocation with precedence constraints." *Defence Science Journal*, vol. 68, no. 2, 2018.

[14] F. Janati, F. Abdollahi, S. S. Ghidary, M. Jannatifar, J. Baltes, and S. Sadeghnejad, "Multi-robot task allocation using clustering method," in *Robot Intelligence Technology and Applications 4: Results from the 4th International Conference on Robot Intelligence Technology and Applications*. Springer, 2016, pp. 233–247.

[15] J. Velagić, D. Delimustafić, and D. Osmanković, "Mobile robot navigation system based on probabilistic road map (prm) with halton sampling of configuration space," in *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*. IEEE, 2014, pp. 1227–1232.

[16] L. Janson, B. Ichter, and M. Pavone, "Deterministic sampling-based motion planning: Optimality, complexity, and performance," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 46–61, 2018.

[17] L.-L. Wang and L.-X. Pan, "Research on sbmpc algorithm for path planning of rescue and detection robot," *Discrete Dynamics in Nature and Society*, vol. 2020, no. 1, p. 7821942, 2020.

[18] L. Ye, J. Chen, and Y. Zhou, "Real-time path planning for robot using op-prm in complex dynamic environment," *Frontiers in Neurorobotics*, vol. 16, p. 910859, 2022.

[19] J. You, J. Jia, X. Pang, J. Wen, Y. Shi, and J. Zeng, "A novel multi-robot task assignment scheme based on a multi-angle k-means clustering algorithm and a two-stage load-balancing strategy," *Electronics*, vol. 12, no. 18, p. 3842, 2023.

[20] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[21] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[22] G. Team, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[23] C. Sun, S. Huang, and D. Pompili, "Llm-based multi-agent reinforcement learning: Current and future directions," *arXiv preprint arXiv:2405.11106*, 2024.

[24] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan, "Autotamp: Autoregressive task and motion planning with llms as translators and checkers," in *2024 IEEE International conference on robotics and automation (ICRA)*. IEEE, 2024, pp. 6695–6702.

[25] J. X. Liu, Z. Yang, B. Schornstein, S. Liang, I. Idrees, S. Tellex, and A. Shah, "Lang2ltl: Translating natural language commands to temporal specification with large language models," in *Workshop on Language and Robotics at CoRL 2022*, 2022.

[26] S. S. Kannan, V. L. Venkatesh, and B.-C. Min, "Smart-llm: Smart multi-agent robot task planning using large language models," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 12 140–12 147.

[27] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[28] P. Li, Z. An, S. Abrar, and L. Zhou, "Large language models for multi-robot systems: A survey," *arXiv preprint arXiv:2502.03814*, 2025.

[29] M. Ahn, M. G. Arenas, M. Bennice, N. Brown, C. Chan, B. David, A. Francis, G. Gonzalez, R. Hessmer, T. Jackson *et al.*, "Vader: Visual affordance detection and error recovery for multi robot human collaboration," *arXiv preprint arXiv:2405.16021*, 2024.

[30] W. Hunt, T. Godfrey, and M. D. Soorati, "Conversational language models for human-in-the-loop multi-robot coordination," *arXiv preprint arXiv:2402.19166*, 2024.

[31] A. Lykov, M. Dronova, N. Naglov, M. Litvinov, S. Satsevich, A. Bazhenov, V. Berman, A. Shcherbak, and D. Tsetserukou, "Llm-mars: Large language model for behavior tree generation and nlp-enhanced dialogue in multi-agent robot systems," *arXiv preprint arXiv:2312.09348*, 2023.

[32] T. Shima, S. J. Rasmussen, A. G. Sparks, and K. M. Passino, "Multiple task assignments for cooperating uninhabited aerial vehicles using genetic algorithms," *Computers & operations research*, vol. 33, no. 11, pp. 3252–3269, 2006.

[33] C. Mouradian, J. Sahoo, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A coalition formation algorithm for multi-robot task allocation in large-scale natural disasters," in *2017 13th international wireless communications and mobile computing conference (IWCMC)*. IEEE, 2017, pp. 1909–1914.

[34] A. Simonetto, J. Monteil, and C. Gambella, "Real-time city-scale ridesharing via linear assignment problems," *Transportation Research Part C: Emerging Technologies*, vol. 101, pp. 208–232, 2019.

[35] E. Bischoff, F. Meyer, J. Inga, and S. Hohmann, "Multi-robot task allocation and scheduling considering cooperative tasks and precedence constraints," in *2020 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE, 2020, pp. 3949–3956.

[36] M. Otte, M. J. Kuhlman, and D. Sofge, "Auctions for multi-robot task allocation in communication limited environments," *Autonomous Robots*, vol. 44, pp. 547–584, 2020.

[37] N. Lindsay, R. K. Buehling, and L. Sun, "A sequential task addition distributed assignment algorithm for multi-robot systems," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 2, p. 51, 2021.

[38] X. Bai, A. Fielbaum, M. Kronmüller, L. Knoedler, and J. Alonso-Mora, "Group-based distributed auction algorithms for multi-robot task assignment," *IEEE Transactions on Automation Science and Engineering*, vol. 20, no. 2, pp. 1292–1303, 2022.

[39] W. Zhao, Q. Meng, and P. W. Chung, "A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario," *IEEE transactions on cybernetics*, vol. 46, no. 4, pp. 902–915, 2015.

[40] A. Agrawal, A. S. Bedi, and D. Manocha, "Rtaw: An attention inspired reinforcement learning method for multi-robot task allocation in warehouse environments," *arXiv preprint arXiv:2209.05738*, 2022.

[41] L. Ratnabala, A. Fedoseev, R. Peter, and D. Tsetserukou, "Magnnet: Multi-agent graph neural network-based efficient task allocation for autonomous vehicles with deep reinforcement learning," *arXiv preprint arXiv:2502.02311*, 2025.

[42] Z. Zhang, X. Jiang, Z. Yang, S. Ma, J. Chen, and W. Sun, "Scalable multi-robot task allocation using graph deep reinforcement learning with graph normalization," *Electronics*, vol. 13, no. 8, p. 1561, 2024.

[43] Z. Chekakta, N. Aouf, S. Govindaraj, F. Polisano, and G. De Cubber, "Towards learning-based distributed task allocation approach for multi-robot system," in *2024 10th International Conference on Automation, Robotics and Applications (ICARA)*. IEEE, 2024, pp. 34–39.

[44] Q. Xu, J. Li, S. Koenig, and H. Ma, "Multi-goal multi-agent pickup and delivery," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 9964–9971.

[45] W. Hönig, S. Kiesel, A. Tinka, J. Durham, and N. Ayanian, "Conflict-based search with optimal task assignment," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2018.

[46] H. Ma, J. Li, T. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," *arXiv preprint arXiv:1705.10868*, 2017.

[47] Z. Zhao, S. Cheng, Y. Ding, Z. Zhou, S. Zhang, D. Xu, and Y. Zhao, "A survey of optimization-based task and motion planning: From classical to learning approaches," *IEEE/ASME Transactions on Mechatronics*, 2024.

[48] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International conference on machine learning*. PMLR, 2022, pp. 9118–9147.

[49] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg, "Text2motion: From natural language instructions to feasible plans," *Autonomous Robots*, vol. 47, no. 8, pp. 1345–1365, 2023.

[50] Y. Xie, C. Yu, T. Zhu, J. Bai, Z. Gong, and H. Soh, "Translating natural language to planning goals with large-language models," *arXiv preprint arXiv:2302.05128*, 2023.

[51] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone, "Llm+ p: Empowering large language models with optimal planning proficiency," *arXiv preprint arXiv:2304.11477*, 2023.

[52] J. Styrud, M. Iovino, M. Norrlöf, M. Björkman, and C. Smith, "Automatic behavior tree expansion with llms for robotic manipulation," *arXiv preprint arXiv:2409.13356*, 2024.

[53] J. X. Liu, A. Shah, G. Konidaris, S. Tellex, and D. Paulius, "Lang2ltl-2: Grounding spatiotemporal navigation commands using large language and vision-language models," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 2325–2332.

[54] D. Müllner, "Modern hierarchical, agglomerative clustering algorithms," *arXiv preprint arXiv:1109.2378*, 2011.

[55] Y. Yuan, P. Yang, H. Jiang, and T. Shi, "A multi-robot task allocation method based on the synergy of the k-means++ algorithm and the particle swarm algorithm," *Biomimetics*, vol. 9, no. 11, p. 694, 2024.

[56] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *Journal of the ACM (JACM)*, vol. 7, no. 4, pp. 326–329, 1960.

[57] P. Gastin and D. Oddoux, "Fast ltl to büchi automata translation," in *International Conference on Computer Aided Verification*. Springer, 2001, pp. 53–65.

[58] J. Ren, H. Miller, K. M. Feigh, S. Coogan, and Y. Zhao, "Ltl-d*: Incrementally optimal replanning for feasible and infeasible tasks in linear temporal logic specifications," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 4495–4502.

[59] NVIDIA, "Nvidia isaac sim," https://developer.nvidia.com/isaac/sim, 2025, accessed: 2025-10-09.

[60] A. Shamsah, Z. Gu, J. Warnke, S. Hutchinson, and Y. Zhao, "Integrated task and motion planning for safe legged navigation in partially observable environments," *IEEE Transactions on Robotics*, vol. 39, no. 6, pp. 4913–4934, 2023.

[61] J. Ren, X. Lin, R. Mineyev, K. M. Feigh, S. Coogan, and Y. Zhao, "Accelerating signal-temporal-logic-based task and motion planning of bipedal navigation using benders decomposition," *arXiv preprint arXiv:2508.13407*, 2025.

[62] L. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of voronoi," *ACM transactions on graphics (TOG)*, vol. 4, no. 2, pp. 74–123, 1985.

[63] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars, *Computational geometry: algorithms and applications*. Springer, 2008.

[64] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[65] R. L. Graham, *Concrete mathematics: a foundation for computer science*. Pearson Education India, 1994.

[66] M. Jünger, G. Reinelt, and G. Rinaldi, "The traveling salesman problem," *Handbooks in operations research and management science*, vol. 7, pp. 225–330, 1995.

[67] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.




**Nan Li** received the M.S. degree in mechanical engineering from the Georgia Institute of Technology in 2026. She is currently pursuing the Ph.D. degree in robotics at the Georgia Institute of Technology under the supervision of Dr. Seung-Kyum Choi and Dr. Ye Zhao. She is with the George W. Woodruff School of Mechanical Engineering. Her research focuses on developing decentralized multi-agent systems that enable efficient and high-quality autonomous coordination and decision-making for multi-robot systems, with applications to multi-robot task allocation, collaborative transport, world models for multi-robot systems, and multi-robot path planning.



**Jiming Ren** received the B.Eng. degree in Mechanical Engineering and Computer Science from the Hong Kong University of Science and Technology, Hong Kong SAR, China, in 2021, and the M.S. degree in Mechanical Engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2023. He is currently pursuing the Ph.D. degree in Robotics at the Georgia Institute of Technology, Atlanta, GA, USA. His research interests include robot motion planning, formal methods, task and motion planning, legged locomotion, and dexterous manipulation. His recent research focuses on learning-based robotic manipulation, vision-language-action models, human demonstration data collection, and the integration of formal planning and learning-based methods for complex robotic tasks.



**Haris Miller** received his B.S. degree in Aerospace Engineering in 2022 and M.S. degree in Robotics in 2025, both from the Georgia Institute of Technology in Atlanta, GA, USA. Haris is currently an Aerospace Engineer working in navigation, guidance, and control (NGC) at Sandia National Laboratory. His research interests include navigation, control, and mechatronics, with a focus on hardware systems, simulation tools, and emulators for aerospace applications.



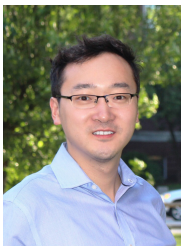
**Samuel Coogan** (Senior Member, IEEE) received the B.S. degree in electrical engineering from the Georgia Institute of Technology, Atlanta, GA, USA in 2010 and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley, Berkeley, CA, USA in 2012 and 2015.

He is currently an associate professor at the Georgia Institute of Technology, Atlanta, GA, USA in the School of Electrical and Computer Engineering. Prior to joining Georgia Tech in 2017, he was an assistant professor at the University of California, Los Angeles from 2015 to 2017.

Prof. Coogan received a CAREER Award from the National Science Foundation in 2018, a Young Investigator Award from the Air Force Office of Scientific Research in 2019, and the Donald P Eckman Award from the American Automatic Control Council in 2020. He is a member of SIAM and a senior member of IEEE.



**Karen M. Feigh** (Senior Member, IEEE) received the B.S. degree in aerospace engineering from Georgia Institute of Technology, the M.Phil. degree in engineering from Cranfield University, and the Ph.D. degree in industrial and systems engineering from Georgia Institute of Technology. She is the David S. Lewis Professor and Associate School Chair for Research in the School of Aerospace Engineering at Georgia Tech. Her research interests include cognitive systems engineering, human factors, human–AI teaming, and decision support for complex sociotechnical systems.



**Ye Zhao** (Senior Member, IEEE) received the Ph.D. degree in mechanical engineering from The University of Texas at Austin, Austin, TX, USA, in 2016. He is currently an Associate Professor with the George W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA, USA. His research interests include robust task and motion planning, reinforcement learning, contact-rich trajectory optimization, and formal methods for legged locomotion, navigation, and dexterous manipulation. Dr. Zhao received the CAREER Award from the National Science Foundation in 2022 and the Young Investigator Award from the Office of Naval Research in 2023.