Agentic Task Allocation for Heterogeneous Multi-Robot Systems: A Hybrid LLM and Local Search Approach

Huibo Zhang
The Hong Kong University of Science and Technology
Shengkang Chen
OceanSurge Robotics LLC
Ziyi Xia
SeamoTech Co. Limited
Huan Yin
The Hong Kong University of Science and Technology
Fumin Zhang
eefumin@ust.hk

The Hong Kong University of Science and Technology

Research Article

Keywords: Task Allocation, Large Language Models, Hybrid Optimization, Multi-Robot Systems, In-Context Learning

Posted Date: June 26th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-8343110/v1

Additional Declarations: No competing interests reported.

Agentic Task Allocation for Heterogeneous Multi-Robot Systems: A Hybrid LLM and Local Search Approach

Huibo Zhang[1], Shengkang Chen[2], Ziyi Xia[3], Huan Yin[1], Fumin Zhang[1*]

[1]Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong Special Administrative Region, China.
[2]OceanSurge Robotics LLC, Altanta, Georgia, USA.
[3]SeamoTech Co. Limited, Hong Kong Special Administrative Region, China.

*Corresponding author(s). E-mail(s): eefumin@ust.hk;
Contributing authors: huibo.zhang@connect.ust.hk;
oceansurgerobotics@gmail.com; mrxiazy@163.com; eehyin@ust.hk;

Abstract

Task allocation among heterogeneous multi-robot teams can be a highly complex NP-hard problem. Traditional approaches, such as Mixed-Integer Programming (MIP) and heuristic search, typically struggle to balance computational scalability with solution quality. To overcome these challenges, we propose an emerging task allocation system based on an intelligent collaboration framework combining a Large Language Model and a Speed-Up Slow-Down(SUSD) derivative-free local search method. Specifically, we treat the Large Language Model as an intelligent reasoning engine, enabling the generation of optimal initial task allocations based on natural language descriptions. Subsequent task allocations are then converged using an iterative local search algorithm. A superior creativity and innovation within our method lie within a memory-guided feedback loop that facilitates utilizing successful historical experiences within a task allocation strategy without requiring parameter updates. Simulation experiments conducted within underwater exploration and urban delivery missions have shown that our task allocation strategy shortens average mission Makespan with 17.07% compared with traditional SUSD methods. At the same time, it surpasses conventional task allocation

methods. Moreover, within our method, we determined that Large Language Model inference times remain relatively stable, thus scalable for large missions.

Keywords: Task Allocation, Large Language Models, Hybrid Optimization, Multi-Robot Systems, In-Context Learning.

1 Introduction

Multi-robot task allocation (MRTA) [1] is a classic multi-agent problem, involving assigning tasks to a set of heterogeneous robots within spatial, temporal, and operational constraints. MRTA problems appear in a range of domains, from underwater surface inspections, aerial deliveries, industrial monitoring systems, and disaster response. Although there have been advances using optimization-based methods [2], market-based methods [3], and heuristics [4], there remain several fundamental challenges. These include representing high-level human intention within task allocation, adaptability within dynamic tasking domains, and reasoning about ambiguous or partially undefined task specifications—the three points that are hard to incorporate using traditional optimization frameworks. Moreover, traditional methods suffer from challenges related to scale and local optima.

Recent breakthroughs in LLMs provide a new paradigm on which these challenges can be tackled. LLMs demonstrate capabilities for contextual reasoning, generalizability on structured prompts, and malleability on dynamic task descriptions, i.e., properties that should be shown by intelligent decision-making agents. By incorporating LLMs within optimization methods, they can act as cognitive-guided planners, understanding task semantics, making use of experience knowledge, and developing well-informed initial allocation strategies. Their compatibility with natural languages makes them more applicable for human-in-the-loop systems, wherein untrained humans provide task goals with high-level task descriptions. Moreover, with refinement methods using feedback, we believe LLMs can make it feasible to improve allocations with a method mix of symbolic knowledge and numerical optimization.

Based on these observations, we develop an optimization strategy with an interactive reasoning mechanism for modeling task assignment. The strategy proceeds as follows. We have an autonomous agent who takes natural language task statements and robot descriptions as input and produces an initial assignment based on priors. To refine these assignments, we use our previous work, SUSD [5], a local search algorithm with no gradients, which refines assignments at every iteration. Moreover, whenever it observes a bottleneck in its optimization loop, it uses in-context learning on its memory buffer without any updates. Fig. 1 shows a visualization of our system, with the LLM serving as the central reasoning unit and reinforced with local numerical search and memory-guided feedback. We will demonstrate our system on a small- and large-scale MRTA scenario with underwater AUV operations and drone deliveries. Our system shows improvements on makespan and convergence rate compared with traditional task allocation algorithms.

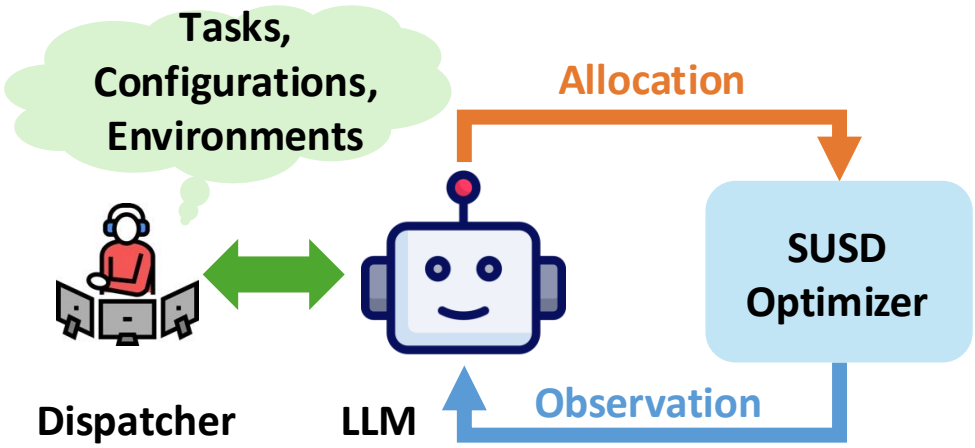

Fig. 1 Illustration of the proposed task allocation method using the agentic approach. A non-expert user poses a query to the system for task allocation among various heterogeneous robots. The system uses an LLM to produce an initial task allocation solution based on observation. Subsequently, an optimization algorithm based on SUSD refines it. The result will be an optimal task allocation solution. It may have the potential capability to achieve efficient task allocation among multiple robots without requiring an expert.

2 Related Work

Task allocation problems on multi-robots have been commonly tackled with optimization methods and market economies [3]. Traditional optimization methods, as exemplified with the Hungarian algorithm [6, 7], and MILP [2, 8, 9], rely on finding global optimal assignments with carefully predefined constraints. Although working well for small or static task graphs, they pose serious scaling problems when confronted with large and dynamic task graphs due to combinatorial complexity [10]

By contrast, market-based methods have provided decentralized and scalable solutions. Robots independently bid on tasks with local utility functions. Examples include MURDOCH [11] and the Consensus Based Bundle Algorithm (CBBA) [12], which have shown resilience against communication break-downs and enabled real-time computation. Moreover, more recent works have attempted a fusion [13] between centralized optimization and market-based heuristics. By contrast, on the opposite side, bio-inspired methods for optimization have emerged as applicable tools within MRTA due to adaptability within unstructured environments. Specifically, among them, it should be noted that algorithm SUSD [14], based on schooling behavior among fish, have been adopted as an alternative local search heuristic devoid of computation as a derivative. It can tackle objective functions with no gradients and deal with noise and local optima. Therefore, it successfully fits within MRTA as an applicable method for real-world scenarios.

In recent years, the emergence of LLMs has introduced new paradigms for reasoning-driven planning in multi-agent systems. Yang *et al.* [15] proposed utilizing

large language models to decompose mission descriptions for autonomous underwater vehicles in underwater exploration scenarios, and assigning task sequences based on their capabilities. Wu *et al.* [16] developed a hierarchical LLM-in-the-loop system that reasons about environmental hazards and dynamically adjusts optimization parameters for performance, safety, and energy efficiency. Obata *et al.* [17] introduced the LiP-LLMs approach, integrating LLM with linear programming to manage task dependencies and improve robustness. The SMART-LLM system [18] similarly leverages LLMs for task decomposition and coalition formation. Outside of robotics, Prieto and García de Soto [19] applied LLMs to construction scheduling, enabling natural language-based planning and resource allocation.

Beyond planning, LLMs have been explored as tools for solving combinatorial optimization problems, including MILP formulation. Li *et al.* [20] proposed a pipeline that extracts decision variables and constraints from natural language to synthesize MILP models with high accuracy. OptiMUS-0.3 [21] enables end-to-end model generation and solving through prompt-based interfaces. LLMOPT [22], Lawless *et al.* [23], and Wang *et al.* [24] further demonstrate LLM applications in solver configuration, cutplane selection, and feasibility improvement through prompt engineering and dynamic temperature scaling.

There have also been some studies on applying LLMs as iterative optimizers. The method for Optimization by PROmpting, or OPRO [25], considers LLMs as an optimizer that produces possible solutions, analyzes them, and improves subsequent solutions with guidance from prompts. It was demonstrated on traditional problems like Linear Regression and the Traveling Salesman Problem (TSP) [26]. The method, FunSearch [27], uses LLMs with external evaluators for finding new solutions for mathematical problems such as Bin Packing and the Cap Set Problem [28].

Some previous works have incorporated LLMs into evolutionary optimization procedures. Liu *et al.* [29], among several others [30–32], have suggested employing LLMs for choosing parent individuals and producing offspring based on crossover and mutation commands. These works demonstrate that LLMs can be utilised as zero-shot optimisers that are capable of learning new instances based on logical reasoning via prompts, instead of relying on domain coding. This work emphasizes the role of LLMs as potentially capable explorers and exploiters within search spaces with high dimensions.

3 Problem Formulation

We address a multi-task–robots single-robot–task (MT–SR) allocation problem in a heterogeneous team. Let $\mathcal{T} = \{t_1, \ldots, t_{n_T}\}$ be the task set and $\mathcal{R} = \{r_1, \ldots, r_{n_R}\}$ the robot set. Each task t_j is defined by its position $\mathbf{q}_j \in \mathbb{R}^2$ and a nominal processing time $\hat{t}_j = t_{\text{base}}$. Each robot r_i has cruising speed v_i and a capability factor $s_{ij} \in \{1, 2\}$ that reflects its efficiency on t_j; the factors are obtained from the capability matrix $\mathbf{Q}$ and the task-type indicator $\mathbf{Y}$ via $\mathbf{S} = \mathbf{Q}^\top \mathbf{Y}$.

Cost Model. Let $\mathbf{A} = [a_{ij}] \in \{0, 1\}^{n_R \times n_T}$ denote the binary assignment matrix, where $a_{ij} = 1$ if task t_j is assigned to robot r_i, and $a_{ij} = 0$ otherwise. Each assignment

incurs two time-based costs:

$$t_{ij}^{E} = \frac{\hat{t}_j}{s_{ij}}, t_{ij}^{T} = \frac{\|\mathbf{p}_i - \mathbf{q}_j\|_2}{v_i} \tag{1}$$

where $\hat{t}_j$ is the nominal duration of task t_j, s_{ij} is the skill coefficient of robot r_i for task t_j, $\mathbf{p}_i$ is the robot's position at the time of assignment, $\mathbf{q}_j$ is the task location, and v_i is the cruising speed of robot r_i.

The total time required for robot r_i to complete all its assigned tasks is given by:

$$T_i = \sum_{j=1}^{n_T} a_{ij} \left(t_{ij}^{E} + t_{ij}^{T} \right). \tag{2}$$

Mission Objective. The optimization goal is to minimize the overall *makespan*, defined as the maximum completion time across all robots:

$$F(\mathbf{A}) = \max_{i \in \mathcal{R}} T_i. \tag{3}$$

in which makespan is preferred over average completion time because: (i) the mission completes only when the last task is done; (ii) it penalizes task imbalance across agents; and (iii) it aligns with industrial MRTA benchmarks.

Optimization Problem. The task allocation is formulated as the following binary combinatorial program:

$$\min_{\mathbf{A}} \quad F(\mathbf{A}) \tag{P}$$

$$\text{s.t.} \quad \sum_{i=1}^{n_R} a_{ij} = 1, \quad \forall j \in \mathcal{T}, \tag{4}$$

$$a_{ij} \in \{0, 1\}, \quad \forall i \in \mathcal{R},\ j \in \mathcal{T}. \tag{5}$$

in which Constraint (4) ensures each task is assigned to exactly one robot, while constraint (5) enforces binary feasibility. Note that Problem (P) is NP-hard. In the following sections, we propose an LLM-guided Speed-Up–Slow-Down method to obtain high-quality task allocations in real-time.

4 Methodology

We put forth a hybrid optimization paradigm with task allocation as a sequential decision-making task, wherein a cognitive guided AI decision-maker tries to improve its task allocation policy incrementally through reasoning based on problem structure and memory-based outcomes. To optimize task allocation and formulate an iterative process that balances exploration and exploitation, we combine a large language model with an SUSD algorithm.

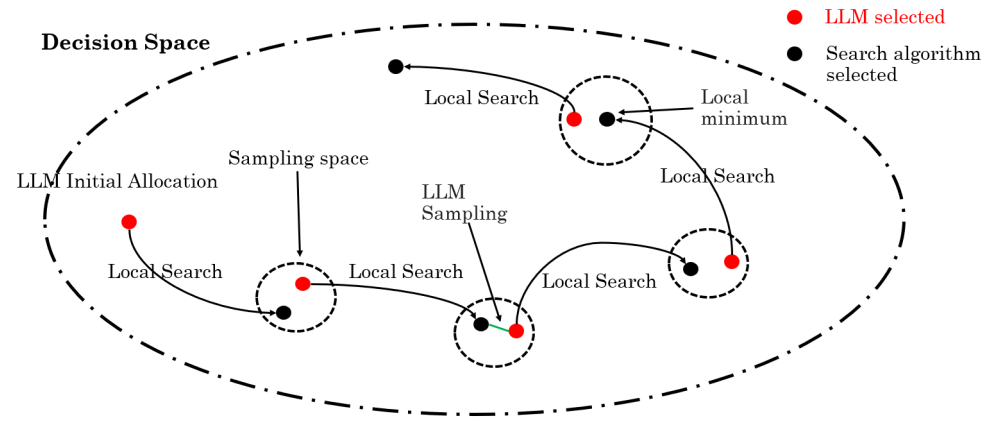

Fig. 2 The hybrid approach using LLM and SUSD. The LLM first generates an initial allocation (red annotation), guiding SUSD to perform gradient-based local search within the decision space (outer ellipse). When SUSD converges to local optima (black dots), the LLM dynamically adjusts the sampling space (inner ellipse shift, red arrow) using memory patterns.This enabled semantic-guided jumps to escape suboptimal regions and achieve global optimization

Specifically, it is designed as a cognitively-guided optimization agent with an iterative refining loop on task allocation based on a complementary combination of an LLM and a local search method. It generates initial solutions based on contextual reasoning, and then uses planning with optimization as a strategy to refine these solutions, and adjusts and updates its strategy based on learning with feedback. The pipeline is shown in Fig. 2.

4.1 Iterative Optimization Loop

The core of the proposed approach is an iterative agentic optimization loop that synergizes symbolic reasoning, probabilistic local refinement, and experience-driven adaptation. This closed-loop architecture enables the agent to progressively refine task allocations based on prior experience, semantic priors, and optimization feedback. The complete procedure is summarized in Algorithm 1.

At each iteration, the agent first generates an initial allocation $\mathbf{A}^{\mathrm{LLM}}(t)$ through LLM-based contextual reasoning, conditioned on the current task set $\mathcal{T}$, robot set $\mathcal{R}$, and memory buffer $\mathcal{H}_{t-1}$. This allocation is then refined by applying the SUSD local search algorithm, resulting in an improved solution $\mathbf{A}^{\mathrm{SUSD}}(t)$. The quality of the refined allocation is evaluated via the mission objective function $F(\mathbf{A}^{\mathrm{SUSD}}(t))$, typically defined as the makespan. The tuple $\left(\mathbf{A}^{\mathrm{LLM}}(t), \mathbf{A}^{\mathrm{SUSD}}(t), S_t\right)$ is stored in the memory buffer $\mathcal{H}_t$ to inform subsequent iterations.

After T iterations, the final solution $\mathbf{A}^*$ is selected as the best-performing allocation observed during the optimization process:

$$\mathbf{A}^* = \arg \min_{\mathbf{A}_i^{\mathrm{SUSD}} \in \mathcal{H}_T} S_i. \tag{6}$$

Algorithm 1 Agentic Optimization with LLM and SUSD

procedure AGENTIC-OPTIMIZATION($\mathcal{T}$, $\mathcal{R}$, f_{LLM}, SUSD)
  Initialize memory buffer $\mathcal{H}_0 \leftarrow \emptyset$
  for $t = 1$ to T **do**
    $\mathbf{A}^{\text{LLM}}(t) \leftarrow f_{\text{LLM}}(\mathcal{T}, \mathcal{R}, \mathcal{H}_{t-1})$
    $\mathbf{A}^{\text{SUSD}}(t) \leftarrow \text{SUSD}(\mathbf{A}^{\text{LLM}}(t))$
    $S_t \leftarrow F(\mathbf{A}^{\text{SUSD}}(t))$
    Update memory buffer: $\mathcal{H}_t \leftarrow \mathcal{H}_{t-1} \cup \left\{ \left(\mathbf{A}^{\text{LLM}}(t), \mathbf{A}^{\text{SUSD}}(t), S_t \right) \right\}$
  end for
  return $\mathbf{A}^* \leftarrow \arg\min_{\mathbf{A}_i^{\text{SUSD}} \in \mathcal{H}_T} S_i$
end procedure

To move forward with exploration beyond local basins of attraction, the agent uses a new peripheral exploration method with structured perturbations introduced within promising allocations. The process allows for an exploration search within neighbor allocations, thus improving the chances of reaching global optimal solutions. As shown in Figure 2, controlled perturbations are made on the best solution obtained so far, followed by SUSD solution refinement and feasibility checks. Only solutions that meet specific constraints will be stored within the memory buffer. Otherwise, a new initialization done through an LLM call continues solution space exploration.

It should be noted that, compared with traditional MRTA solver architectures or heuristic pipeline architectures with fixed heuristic steps, the new agentic framework bridges symbolic reasoning based on LLM-generated task allocations, numerical local search via SUSD, and memory-guided experiential learning as constituent components within an interpretable procedural loop.

4.2 LLM-Based Allocation Initialization

The decision cycle commences with the generation of an initial task allocation, denoted as $\mathbf{A}^{\text{LLM}}$, via a Large Language Model (LLM). The LLM functions as a policy function:

$$\mathbf{A}^{\text{LLM}} = f_{\text{LLM}}(\mathcal{T}, \mathcal{R}, \mathcal{P}) \tag{7}$$

transforming the natural language context—comprising task set $\mathcal{T}$, robot set $\mathcal{R}$, and positional information $\mathcal{P}$—into a feasible allocation matrix satisfying all problem constraints.

To effectively integrate the continuously spatial domain associated with multi-robot task allocation problems with the discretely represented token space modeled by large language models, we employ a coordinate-based serialization method. As opposed to a quadratic scaling Euclidean distance matrix requiring ($O(N^2)$) computations, we serialize Cartesian coordinates (x, y) directly in the input. It is recognized that, while large language models clearly demonstrate emergent capabilities within spatial reasoning domains (such as finding clusters), these models will never possess a level of precision necessary for optimal distance minimization and will, more often, assign coordinates as semantic tokens instead. Moreover, this particular shortcoming of LLMs

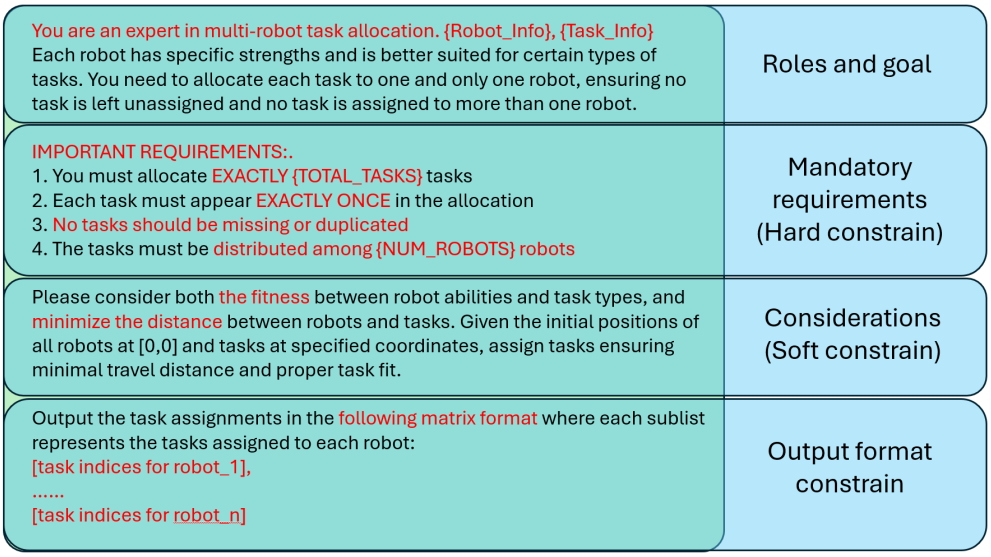

Fig. 3 A case study of LLM-based Initialization prompt. We leverage LLM to analyze the natural language from human beings and provide a result for MRTA solving.

becomes an explicit design principle within our solution framework: LLMs are solely tasked with providing a high-level semantic valid "warm start" solution reflecting capability-based matches and regional notions, with specific numerical optimization relegated to SUSD local search.

To steer the current generation, we employ a structured prompting approach with four fundamental elements: (1) Role Definition, presenting the model as a domain specialist; (2) Hard Constraints, addressing feasibility issues like task repetition; (3) Soft Constraints, encouraging robot and task compatibility and closeness; and (4) Formatting Guidelines, demanding a JSON-like structure consumable for direct parsing. A sample cut-out of our structured prompting technique is shown below in Fig. 3. Our structured initialization technique achieves a strong contextual prior and significantly boosts convergence progress within the next optimization stage.

4.3 SUSD-Based Probabilistic Refinement

The second stage of the agent's decision cycle performs local refinement on the LLM-generated assignment using the SUSD algorithm—a bio-inspired, derivative-free optimization method that leverages probabilistic motion in principal search directions. Following the relation process in [13], SUSD models the allocation matrix as a distribution parameterized by $\mathbf{\Theta} \in \mathbb{R}^{n_R \times n_T}$, where each element Θ_{ij} determines the likelihood of assigning task t_j to robot r_i:

$$p_{ij} = \frac{\exp(\Theta_{ij})}{\sum_{k=1}^{n_R} \exp(\Theta_{kj})}. \tag{8}$$

These probabilities are discretized to produce a binary assignment matrix $\mathbf{A}^{\text{SUSD}}$ via sampling and thresholding. The SUSD algorithm then iteratively updates $\mathbf{\Theta}$ using a velocity-based feedback rule:

$$\mathbf{\Theta}^{(t+1)} = \mathbf{\Theta}^{(t)} + \eta \mathbf{z}^{(t)} \mathbf{n}^{(t)} + \mathbf{u}_f^{(t)}, \tag{9}$$

where η is a step-size constant, $\mathbf{z}^{(t)}$ reflects directional fitness feedback, $\mathbf{n}^{(t)}$ is a principal direction computed via PCA of the sampled assignment distributions, and $\mathbf{u}_f^{(t)}$ modulates search acceleration. This formulation exploits the low-dimensional structure of agent trajectories and performs guided exploration without access to gradients.

Following the convergence analysis in [5], the SUSD algorithm asymptotically aligns its principal search direction with the steepest descent direction of the objective function, enabling effective convergence to local minima. For an objective function $F(\mathbf{A})$ that is twice continuously differentiable, and under standard smoothness conditions, the dynamics of SUSD satisfy:

$$\dot{\bar{F}} = \gamma \bar{S} \cdot \langle \nabla F, v_1 \rangle \,, \tag{10}$$

where $\dot{\bar{F}}$ denotes the time derivative of the average objective value, $\gamma \in \mathbb{R}_+$ is a system gain, $\bar{S}$ is the mean fitness score across the population, ∇F is the gradient of the objective, and v_1 is the principal eigenvector of the agent trajectory covariance matrix. The term $\langle \nabla F, v_1 \rangle$ represents the directional derivative along the search direction.

The evolution of the search direction v_1 is further characterized by:

$$\dot{v}_1 = -\sum_{k=2}^{d} \frac{\lambda_k}{\lambda_k - \lambda_1} v_k v_k^\top \nabla F + \omega, \tag{11}$$

where λ_k and v_k denote the eigenvalues and eigenvectors of the covariance matrix, respectively, and ω is a bounded perturbation from higher-order system terms.

This formulation implies that v_1 is continuously adjusted toward the negative gradient direction, particularly when the trajectory distribution is sharply peaked ($\lambda_1 \gg \lambda_k$ for $k > 1$).

Finally, under an exponential transformation of the objective (used to regularize scale), SUSD satisfies input-to-state stability (ISS), ensuring convergence to a local optimum:

$$F(\mathbf{A}^*) \leq F(\mathbf{A}^{\text{LLM}}), \tag{12}$$

where $\mathbf{A}^*$ is the best assignment found by SUSD and $\mathbf{A}^{\text{LLM}}$ is the LLM-generated initialization. This guarantees that the search process always improves or preserves solution quality relative to the initial LLM output.

4.4 Memory-Guided Refinement

To enable adaptive refinement and memory-guided decision-making, we introduce a feedback-conditioned prompt that incorporates historical optimization results. As shown in Fig. 4, this second-stage prompt explicitly provides the LLM with a compact summary of recent allocations and their performance. The LLM is then instructed to

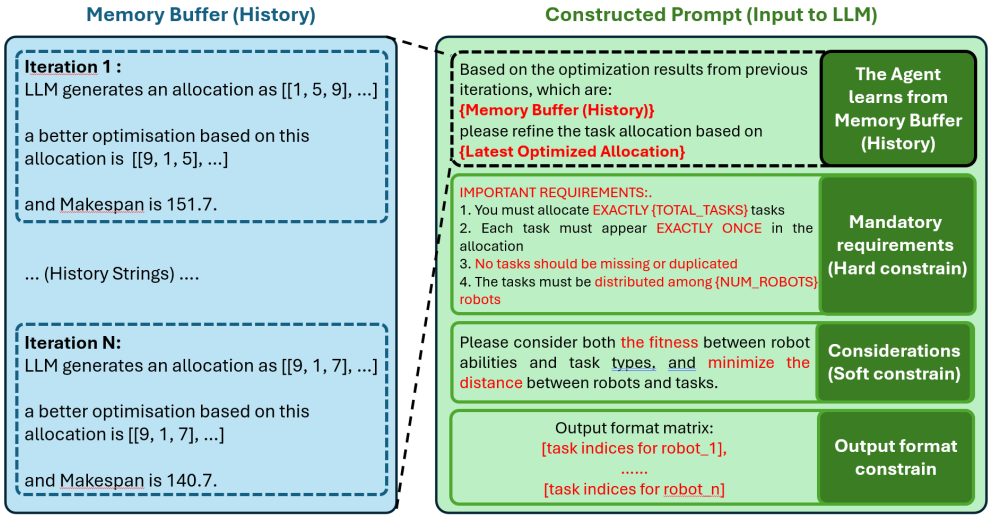

Fig. 4 Anatomy of the memory-guided feedback prompt. The figure illustrates the data flow from the historical experience store to the active reasoning context. **(Left)** The *memory buffer* $\mathcal{H}_t$ stores structured tuples of previous attempts, recording both the initial LLM proposal A^{LLM} and the SUSD-refined solution A^{SUSD} together with their performance. **(Right)** The *constructed prompt* dynamically injects these records as serialized textual feedback. By explicitly juxtaposing the LLM's original proposal against the "better optimization" found by SUSD (highlighted in the injection block), the agent performs in-context learning to identify and correct spatial or logical inefficiencies in subsequent iterations.

refine the allocation based on these recent experiences, while strictly respecting all mandatory assignment constraints.

4.4.1 Experience Buffer as an External Memory

To enable experience-based learning, the agent maintains a memory buffer $\mathcal{H}_t$ containing records of prior task allocations, their refined solutions, and corresponding performance scores:

$$\mathcal{H}_t = \left\{ \left(A_i^{\text{LLM}}, A_i^{\text{SUSD}}, S_i \right) \right\}_{i=1}^{t}, \tag{13}$$

where S_i denotes the makespan associated with the refined solution A_i^{SUSD}. Each tuple therefore records (i) the LLM's initial proposal, (ii) the locally optimized allocation produced by SUSD, and (iii) the resulting task-completion cost. This buffer serves as a lightweight, dynamically updated external memory that accumulates empirical evidence about which allocation patterns tend to perform well under different task and robot configurations.

Whenever a new optimization episode is completed, the corresponding tuple is appended to $\mathcal{H}_t$. Optionally, the buffer can be truncated to a fixed maximum size by discarding the oldest entries, which keeps the memory focused on recent experience and prevents unbounded growth.

4.4.2 Retrieval Mechanism and Top-n Selection

Retrieval mechanism: To prioritize relevant experience without exceeding the token limit, we implement a focused retrieval strategy. Conceptually, this operates analogously to an attention mechanism: the current optimization goal and task context act as a *query*, while historical high-quality solutions stored in the buffer act as *keys* and *values*. Instead of performing explicit high-dimensional vector attention, we adopt a simple yet effective ranking scheme based on performance.

Concretely, for the current iteration we rank all stored records in $\mathcal{H}_t$ according to their makespan S_i:

$$\big(A_{(1)}^{\mathrm{LLM}}, A_{(1)}^{\mathrm{SUSD}}, S_{(1)}\big), \ldots, \big(A_{(t)}^{\mathrm{LLM}}, A_{(t)}^{\mathrm{SUSD}}, S_{(t)}\big), \tag{14}$$

with $S_{(1)} \le S_{(2)} \le \cdots \le S_{(t)}$. We then retrieve the top-n records

$$\mathrm{Retrieve}(\mathcal{H}_t) = \big\{\big(A_{(k)}^{\mathrm{LLM}}, A_{(k)}^{\mathrm{SUSD}}, S_{(k)}\big)\big\}_{k=1}^{n}, \tag{15}$$

where n is a tunable hyperparameter whose influence is analyzed in the memory ablation study. In practice, n is chosen such that the serialized text describing these records fits comfortably within the context window of the LLM.

This top-n selection ensures that the LLM conditions its next decision on "proven" high-quality patterns, rather than noisy or suboptimal ones. At the same time, using multiple records (instead of only the single best example) exposes the model to slightly different yet still effective allocation strategies, which empirically improves robustness and generalization.

4.4.3 Feedback-Conditioned Prompt Construction

The retrieved tuples are then serialized into natural-language feedback and injected into the prompt as an additional context block. Each record is presented in a structured "before/after" form: the initial LLM allocation A_i^{LLM}; the improved SUSD allocation A_i^{SUSD}, and the associated makespan S_i.

The feedback block explicitly highlights how the SUSD-refined solution improves upon the original proposal (e.g., by reducing long travel segments, balancing the number of tasks per robot, or grouping spatially nearby tasks), and instructs the LLM to imitate such improvements when generating the next allocation.

Let $(\mathcal{T}, \mathcal{R})$ denote the current task set and robot set. At iteration $t+1$, the LLM generates a new candidate solution A_{t+1}^{LLM} by conditioning on both the current problem context and the retrieved memory:

$$A_{t+1}^{\mathrm{LLM}} = f_{\mathrm{LLM}}\big(\mathcal{T}, \mathcal{R}, \mathrm{Retrieve}(\mathcal{H}_t)\big), \tag{16}$$

where $f_{\mathrm{LLM}}(\cdot)$ denotes the mapping implemented by the language model under the given system and user instructions. Importantly, all mandatory assignment constraints (e.g., each task must be assigned exactly once, and only to feasible robots) are restated in the prompt and enforced during post-processing, ensuring that the LLM's output remains a valid allocation.

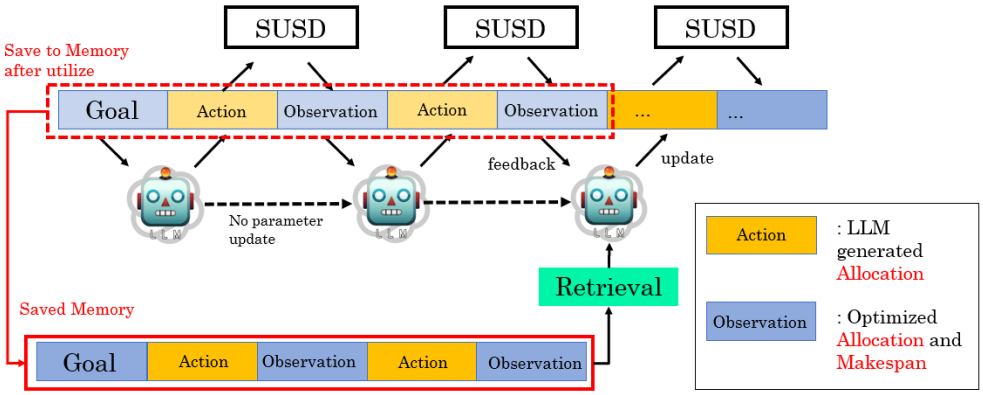

Fig. 5 Iterative optimization and memory update loop. At each iteration, the LLM proposes a new allocation conditioned on the current context and retrieved memory. SUSD then refines this proposal and evaluates its makespan. The resulting optimized solution and feedback are stored in the memory buffer for future retrieval, enabling the agent to update its behavior via in-context learning without any parameter updates.

4.4.4 Iterative Optimization Loop as a Learning Process

The resulting allocation A_{t+1}^{LLM} is passed to SUSD, which performs local search in the continuous relaxation space and returns a refined allocation A_{t+1}^{SUSD} with evaluated makespan S_{t+1}. The new tuple $\left(A_{t+1}^{\text{LLM}}, A_{t+1}^{\text{SUSD}}, S_{t+1}\right)$ is then added back to the memory buffer, closing the feedback loop (Fig. 5).

From a learning perspective, this architecture simulates an agent that refines its heuristic policy over time based on empirical feedback, while keeping the LLM parameters frozen. The memory buffer thus acts as a fast, task-specific adaptation mechanism: instead of retraining or fine-tuning the model, the agent updates only its external experience store and the corresponding prompts. This design maintains the stability and safety of a pretrained model, yet still achieves continual improvement across iterations and scenarios.

5 Experiments

The experimental evaluation is designed to assess the effectiveness, robustness, and scalability of the proposed LLM-SUSD agentic optimization approach for MRTA.

5.1 Experimental Setup

5.1.1 Instance generation

Two application scenarios with different scales and heterogeneity are considered. For each scenario, **six** problem instances are generated via Latin Hypercube Sampling (LHS) to ensure diverse spatial and semantic task distributions regarding task locations, types, and robot counts. Each instance is solved **20 independent times** to average out the stochasticity of both the LLM initialization and the SUSD optimizer.

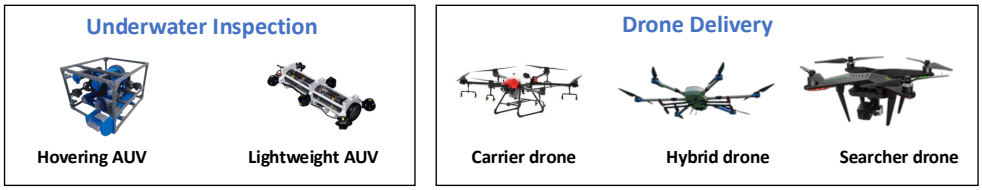

Fig. 6 Robots for experimental validation. Two classical scenarios are established: AUVs are employed for inspection tasks in underwater environments, and drones are deployed for urban delivery.

5.1.2 Scenario 1: Small-scale underwater inspection

A 100×100 m 2-D area contains 10–20 tasks, each labeled either *survey* or *inspection*. The robot team comprises 30–50 Autonomous Underwater Vehicles (AUVs), all starting from the origin point $(0, 0)$. The team is composed of two types of AUVs:

- **Hovering AUV** (precise station keeping for close-range inspection), cruising speed $v = 2\,\mathrm{m\,s^{-1}}$;
- **Lightweight AUV** (large-area coverage for survey), cruising speed $v = 2\,\mathrm{m\,s^{-1}}$.

Although both share the same speed, they differ in task efficiency:

$$\mathbf{Q}_{\mathrm{AUV}} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix},$$

where column 1 corresponds to Hovering AUVs and column 2 to Lightweight AUVs.

5.1.3 Scenario 2: Large-scale urban drone delivery

A 300×300 m urban grid hosts 30–50 delivery tasks clustered around high-rise buildings. The team consists of 8–10 UAVs, all starting from the depot at the origin $(0, 0)$. The UAVs are randomly selected from the following three types:

- **Searcher drones** (long-range reconnaissance), cruising speed $v = 20\,\mathrm{m\,s^{-1}}$;
- **Carrier drones** (short-range cargo transport), cruising speed $v = 10\,\mathrm{m\,s^{-1}}$;
- **Hybrid drones** (balanced platform), cruising speed $v = 15\,\mathrm{m\,s^{-1}}$.

Tasks are labelled as either *search* or *delivery*. The capability matrix is:

$$\mathbf{Q}_{\mathrm{UAV}} = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \end{bmatrix},$$

where the first row corresponds to search tasks and the second row to delivery tasks.

5.1.4 LLM and optimizer settings

All prompts are submitted to the `gpt-4-0613` API[1] with temperature 0.7 and a token limit of 1500. No fine-tuning is performed; model calls are purely zero-shot. The derivative-free SUSD optimiser is run for $T_{\max} = 100$ iterations with primary step gain $k_1 = 2.0$, secondary gain $k_2 = 1.0$ ($k_2 = 0.5k_1$), desired dispersion $d_{\text{des}} = 0.2$, exploration noise scale $\varepsilon = 0.2$, and population size $M = 2n_R * n_T$. Hyper-parameters are fixed across all experiments.

5.2 Comparison Methods

We compare the proposed approach, named LLM+SUSD, against a diverse suite of baseline methods:

- **Market-Based Task Allocation:** A decentralized auction mechanism focusing on distributed decision-making but lacking global coordination.
- **SUSD-Based Task Allocation:** Derivative-free local search initialized with a random allocation matrix, testing performance without semantic priors.
- **Solver (Branch-and-Bound):** A centralized MILP solver based on the CBC library. It finds global optima but is limited to small-scale instances (< 20 tasks).
- **Brute-Force Search:** Exhaustive enumeration used only for validation on micro-scale problems (< 10 tasks).
- **Genetic Algorithm (GA):** An evolutionary metaheuristic (population=50, uniform crossover=0.6, adaptive mutation=0.2).
- **Particle Swarm Optimization (PSO):** Swarm-based method (particles=100, $c_1 = 0.7, c_2 = 0.9, w = 0.7$).

5.3 Overall Comparative Performance

We first evaluate the proposed LLM-guided optimization approach against the six baseline methods. Table 1 summarizes the makespan improvement percentages across twelve task allocation cases. The improvement metric is defined as:

$$\text{Improvement } (\%) = \frac{M_{\text{baseline}} - M_{\text{proposed}}}{M_{\text{baseline}}} \times 100\% \tag{17}$$

where M_{baseline} and M_{proposed} denote the average makespan of the baseline and our method, respectively.

On average, the proposed approach achieves a 21.64% reduction in makespan compared to PSO, 17.07% compared to SUSD with random initialization, 10.22% compared to GA, and 8.40% compared to market-based allocation. Even in small-scale scenarios where MILP solvers yield exact solutions, our method achieves a 13.12% average improvement over solver-based allocations (likely due to solver timeouts or heuristic cutoffs), highlighting its ability to approach near-optimality efficiently.

Fig. 7 and Fig. 8 illustrate the performance distribution. In small-scale problems, the proposed approach consistently attains near-optimal solutions, outperforming

[1] OpenAI, 2024-06-13 release.

Table 1 Makespan Improvement (%) Compared to Other Task Allocation Methods Across All Cases
(- represents methods exceeding time limits)

Case	GA	PSO	Market-based	SUSD-based	Solver	Optimality gap
3 Robot 10 Tasks	2.03	13.77	5.50	2.12	11.20	-0.23
3 Robot 18 Tasks	2.97	14.17	7.93	8.41	11.44	—
4 Robot 12 Tasks	9.06	17.05	8.97	12.68	11.51	—
4 Robot 20 Tasks	7.16	18.53	6.16	13.72	8.25	—
5 Robot 15 Tasks	8.25	15.79	9.25	12.56	20.17	—
5 Robot 16 Tasks	7.63	17.59	10.03	14.29	16.16	—
8 Robot 30 Tasks	9.63	23.77	1.31	20.84	—	—
8 Robot 50 Tasks	8.63	26.25	2.3	16.37	—	—
9 Robot 40 Tasks	10.56	12.82	14.12	16.66	—	—
9 Robot 45 Tasks	24.87	39.36	15.56	31.23	—	—
10 Robot 32 Tasks	19.51	32.10	12.09	29.49	—	—
10 Robot 35 Tasks	15.33	28.58	7.53	25.84	—	—
Average	**10.22**	**21.64**	**8.40**	**17.07**	**13.12**	**-0.23**

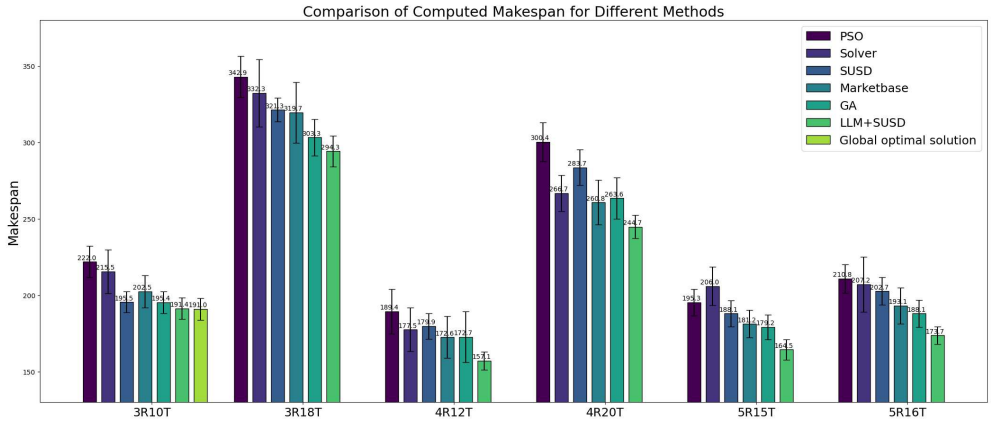

Fig. 7 Computed makespan across baseline methods for smaller-scale MRTA problems. The proposed approach achieves near-optimality in all cases.

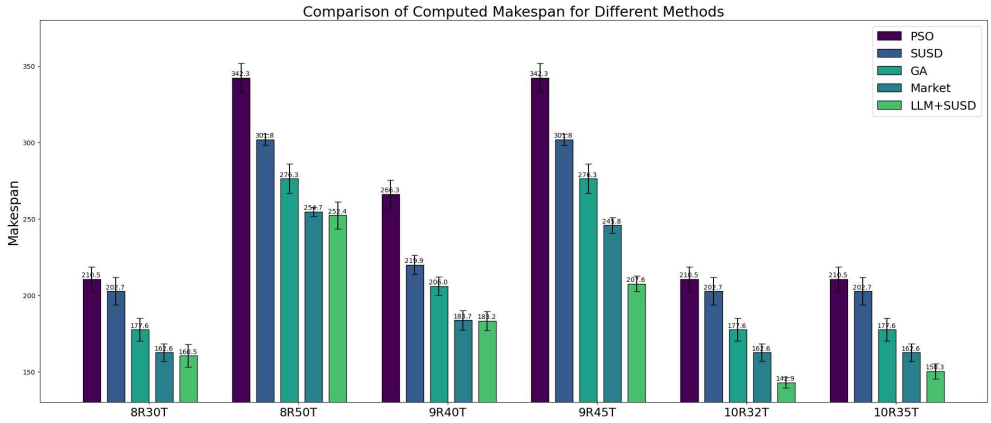

Fig. 8 Comparison of computed makespan for larger-scale problems. Our approach shows strong scalability and outperforms other heuristics.

heuristics. In large-scale settings where exact solvers fail, our approach maintains strong scalability, achieving the lowest average makespan while competitors like PSO suffer from premature convergence.

5.4 Ablation Studies on Agentic Mechanisms

5.4.1 Impact of Initialization Strategy

To evaluate the role of semantic priors, we compare our LLM-based initialization against three randomized sampling strategies followed by SUSD:

- **Random (Redistribution):** Uniform random reallocation.

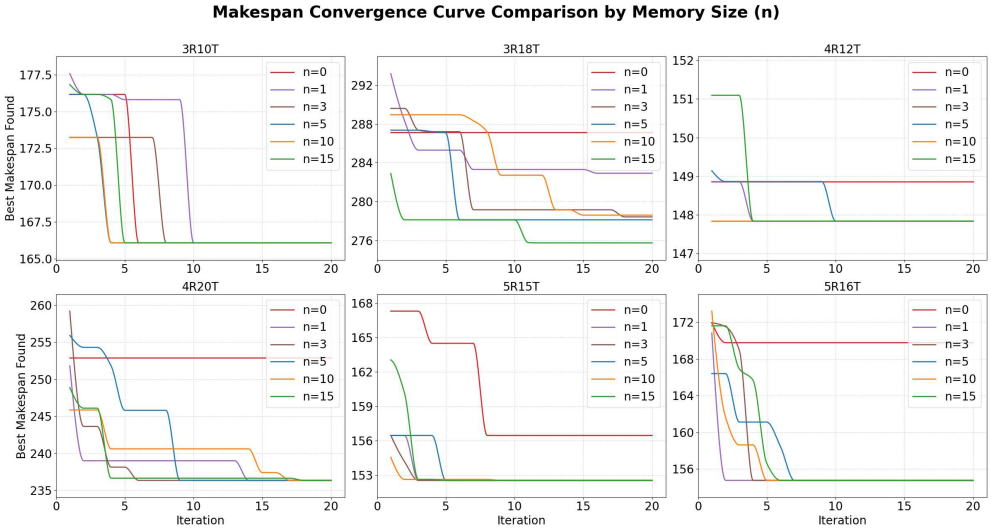

Fig. 9 Makespan Convergence Curve Comparison by Memory Size. Larger memory buffers correlate with faster convergence and better solutions, with diminishing returns after $n = 5$.

- **Random (Task Swap):** Randomly exchanging one task between two robots.
- **Random (Group Swap):** Randomly swapping entire task sets between two robots.

Comparison results (detailed in the discussion of convergence below) indicate that LLM-based initialization consistently provides a superior starting point (warm start), leading to faster convergence and avoiding poor local minima common in coarse random initializations.

5.4.2 Impact of Memory Buffer Size

To quantitatively verify the effectiveness of the memory-guided feedback mechanism, we conducted a sensitivity analysis on the memory buffer size $n \in \{0, 1, 3, 5, 10, 15\}$ on six representative test cases.

- **Baseline ($n = 0$):** No memory feedback; effectively independent zero-shot attempts.
- **Proposed Method ($n > 0$):** Retrieves top-n historical solutions to guide reasoning.

Fig. 9 illustrates the convergence curves. The baseline ($n = 0$) consistently exhibits the poorest performance, often stagnating. In contrast, enabling even minimal memory ($n = 1$) yields significant improvement. While larger buffers improve performance, the marginal gain decreases as n increases, suggesting $n = 5$ strikes an optimal balance between quality and efficiency.

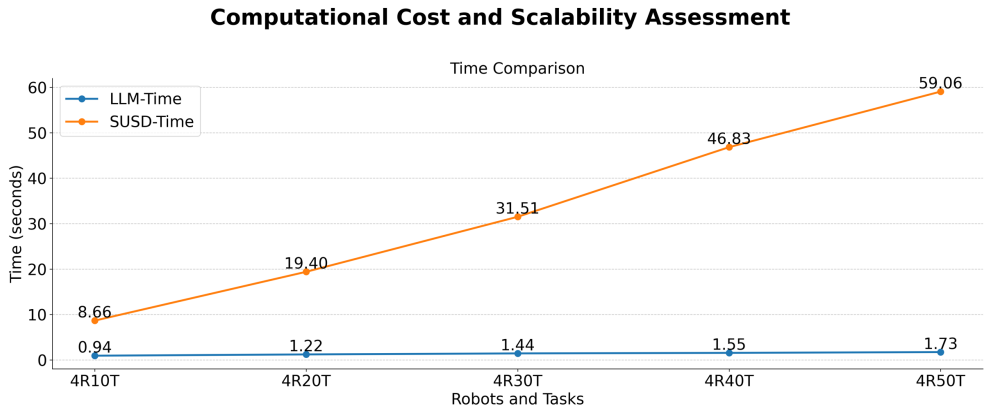

Fig. 10 Computational time breakdown. LLM inference time (T_{LLM}) remains flat, while optimization time (T_{SUSD}) grows linearly.

5.5 Scalability and Efficiency Analysis

5.5.1 Computational Cost Assessment

To identify computational bottlenecks, we decompose execution time: $T_{total} = T_{LLM} + T_{SUSD}$. We conducted a *Task Scalability* experiment (4 robots, task count $N_T \in \{10, \ldots, 50\}$).

As shown in Fig. 10, T_{LLM} exhibits a remarkably flat profile (0.94s to 1.73s) despite a fivefold increase in tasks, indicating LLM inference is insensitive to linear input growth within the context window. Conversely, T_{SUSD} grows linearly (8.66s to 59.06s), dominating the computation (97% at 4R50T). This identifies numerical optimization, not the LLM, as the primary scaling bottleneck.

5.5.2 Convergence and Early Optimal Discovery

We compare the convergence behavior of LLM+SUSD against 12 benchmark scenarios. The following convergence trends, as shown in Fig. 11, reveal faster, smooth convergence for LLM+SUSD with reduced variance compared to random baselines.

Secondly, we examine the Priority Success Rate, defined as the ratio of runs where a method is the first to find a near-optimal solution (±1% of global minimum). As depicted in Fig. 12, LLM guided refinement is leading in all such cases. This again verifies that semantic priors put the optimizer from the very beginning in relevant regions of the search space, making it win the race to optimality.

5.6 Qualitative Trajectory Analysis

To better illustrate the optimization process, we trace out the entire trajectory with 4 robots and 20 tasks (Fig. 13), with different colors signifying optimal task assignments for respective robots. Beginning with LLM initialization, we see that the assignment is random, with multiple robots often performing tasks outside their optimal regions

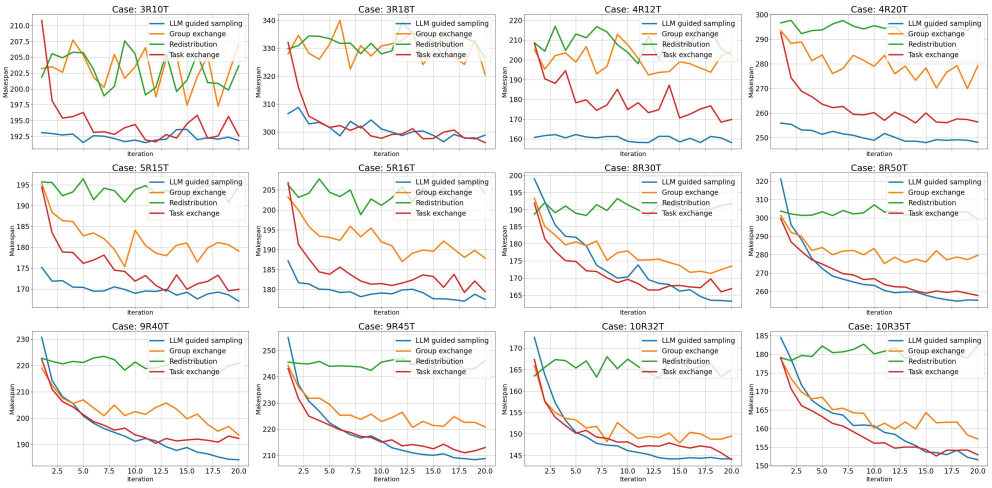

Fig. 11 Convergence trends for the proposed approach on 12 benchmark tasks. LLM-guided sampling improves makespan and reduces performance variance.

marked by variable-color regions. As we trace out the assignments that result from iterative refinement, we see that the assignment increasingly approaches an optimal assignment with all robots tending towards optimal task performance. The end result culminates with an optimal assignment, with task assignments perfectly complementary with ground truth colors, indicating optimal task assignment for each robot.

6 Discussion

The proposed solution shows satisfactory performance capabilities on varied task assignment problems, but some implications and limitations associated with it and the computational complexities involved are worth mentioning.

6.1 Scalability and Computational Bottlenecks

Although these scaling methods favor scaling with regards to solution quality, our cost scaling analysis shows different scaling methods. Defying conventional thinking that Large Language Models bring about cumbersome delay, our experiments, as shown in Fig. 10, show that LLM inference delay is no longer an issue as task complexity escalates. The main CPU bottleneck for our method arises from its iterative numerical refinement step, denoted as T_{SUSD}. As shown, it accounts for a big chunk of overall computation time within large scaling instances, including 97% within 4R50T. It should be noted here that our current implementation of SUSD per se is sequential. As a global search optimization technique based on populations and thus a derivative-free optimization technique. Its implementation on GPUs would significantly lower costs associated with T_{SUSD}, lowering costs substantially and potentially making it operate within large swarms.

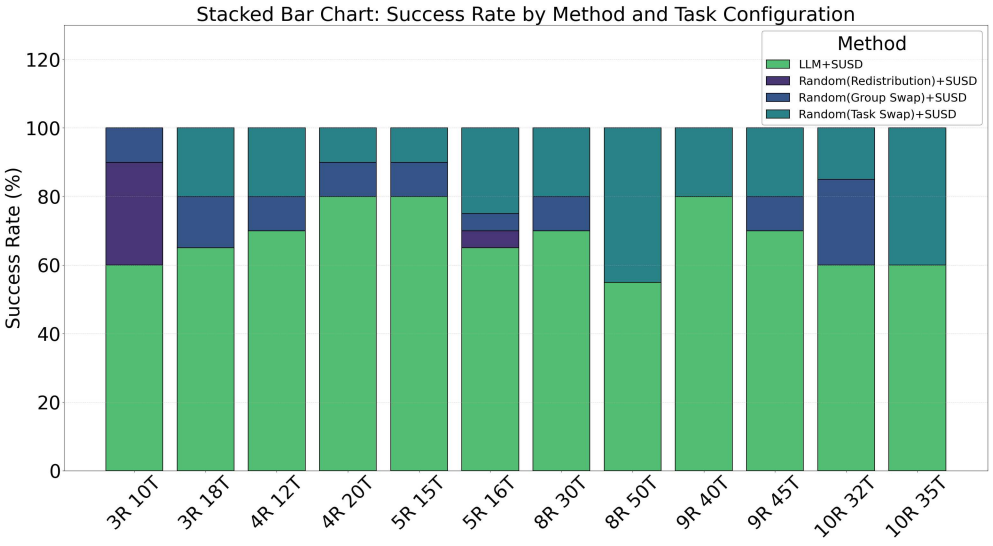

Fig. 12 Percentage of runs where each method is the first to reach a near-optimal solution. LLM+SUSD achieves dominant early-discovery performance.

6.2 Robustness and Generalization Limits

Despite the strength of generalizability that the approach exhibits under zero-shot instances, it presently requires carefully structured prompts about operations on matrices. It might have ambiguous task representation or variability within robot metadata, potentially propagating error at initialization steps. Secondly, as it allows short-term learning and as shown in ablation on memory, it does not have persistent learning. The agent "resets" among disparate missions. It might be useful in learning successful approaches via utilization of reinforcement learning methods and vector databases.

6.3 Real-World Deployment Considerations

As far as applicative usage goes, and more specifically in relation to a heterogeneous set of vehicles as seen with the AUVs and UAVs assessed, there have been some challenges from an engineering perspective. At an abstract level, there have been considerations based on reliable communications. As an AUV and with an acoustic link that spans an underwater environment, there might be some reliability weakness with regards to having a centralized LLM. Thus, an indication towards a distributed platform would be more optimal with local reallocs at a lighter level. Also, safety verification at runtime would be preferred.

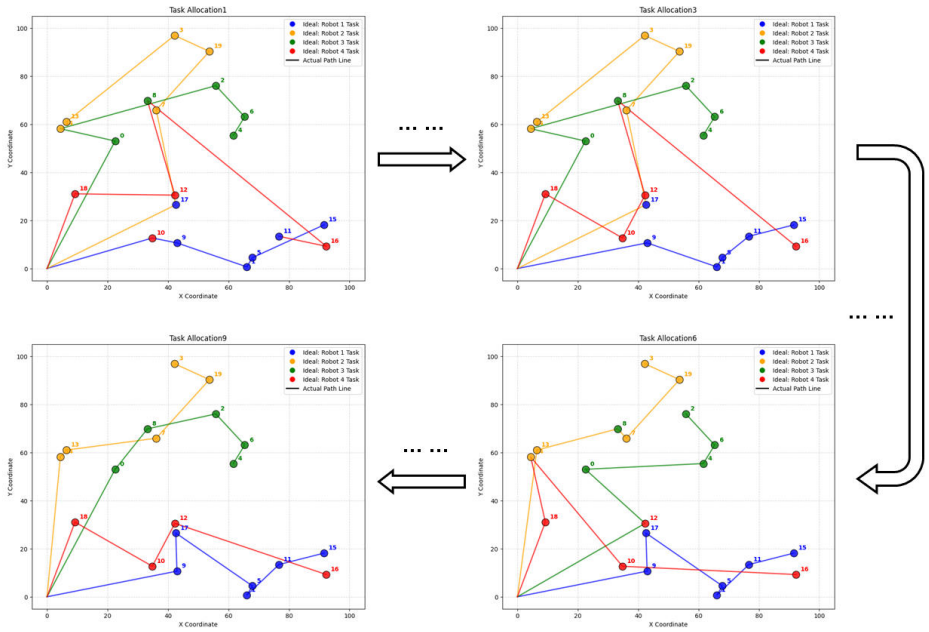

Fig. 13 Trajectory evolution: starting from the LLM-initialized baseline, evolving through intermediate iterative refinements, and culminating in the final solution. The progression demonstrates the continuous optimization of task grouping and spatial efficiency.

7 Conclusion and Future Work

In this article, an original framework for task allocation among Multi-Robot Systems that integrates the semantic reasoning capabilities of Large Language Models with the accuracy offered by local, derivative-free optimization methods (SUSD) will be introduced. To achieve this, LLM will be incorporated within an iterative learning loop facilitated by a memory buffer.

The three major findings that emerge from our comprehensive empirical analysis are these:

It can be seen that the hybrid method always performs better than traditional solvers (MILP) and metaheuristics (GA, PSO) on average with a percentage difference of 17.07% against traditional SUSD. The proposed memory-guided feedback strategy greatly enhances convergence speed. Ablation experiments demonstrate that it is sufficient to retain some memory about successful improvements for efficient escape from local optima.

Computational analysis reveals that the LLM module has very high scaling capabilities with almost zero latency overhead, thus justifying the applicability of large models as 'cognitive planners' within the loop. The focus of future research will be on three fronts: (1) overcome the numerical bottleneck challenge with parallelization of the SUSD algorithm, (2) work on "LLM on the Edge" approaches with a decentralized

focus on lessening dependence on cloud computing, and (3) incorporate evolutionary algorithms that will allow for adaptability on long-term tasks with dynamic changes. The applicability of this neuro-symbolic model will allow researchers to tackle more combinatorial problems involving domains like logistics scheduling and multi-agent path finding.

Declarations

- Ethics approval and consent to participate: Not applicable
- Consent for publication: Not applicable
- Competing interests: The authors declare that they have no competing interests.
- Funding: The work described in this paper was partially supported by grants AoE/E-601/24-N, 16203223, and C6029-23G from the Research Grants Council of the Hong Kong Special Administrative Region, China.
- Authors' contributions: H. Zhang contributed to conceptualization, methodology, algorithm development, software, validation, formal analysis, investigation, data curation, writing—original draft, and writing—review and editing. S. Chen contributed to methodology (SUSD integration), software (simulation platform), validation, formal analysis, and investigation. Z. Xia contributed to software (LLM integration and prompt engineering), formal analysis, and writing—review and editing. H. Yin contributed to supervision, project administration, resources, and writing—review and editing. F. Zhang contributed to supervision, funding acquisition, conceptualization (theoretical framework), and writing—review and editing. All authors read and approved the final manuscript.
- Acknowledgements: The authors would like to thank the anonymous reviewers for their constructive comments and suggestions, which helped improve the quality of this paper.

References

[1] Korsah, G.A., Stentz, A., Dias, M.B.: A comprehensive taxonomy for multi-robot task allocation. The International Journal of Robotics Research **32**(12), 1495–1512 (2013)

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

[3] Dias, M.B., Zlot, R., Kalra, N., Stentz, A.: Market-based multirobot coordination: A survey and analysis. Proceedings of the IEEE **94**(7), 1257–1270 (2006)

[4] Khamis, A., Hussein, A., Elmogy, A.: Multi-robot task allocation: A review of the state-of-the-art. Cooperative robots and sensor networks 2015, 31–51 (2015)

[5] Al-Abri, S., Lin, T.X., Tao, M., Zhang, F.: A derivative-free optimization method with application to functions with exploding and vanishing gradients. IEEE

Control Systems Letters **5**(2), 587–592 (2020)

[6] Chopra, S., Notarstefano, G., Rice, M., Egerstedt, M.: A distributed version of the hungarian method for multirobot assignment. IEEE Transactions on Robotics **33**(4), 932–947 (2017)

[7] Kuhn, H.W.: The hungarian method for the assignment problem. Naval research logistics quarterly **2**(1-2), 83–97 (1955)

[8] Atay, N., Bayazit, B.: Mixed-integer linear programming solution to multi-robot task allocation problem (2006)

[9] Falsone, A., Margellos, K., Prandini, M.: A decentralized approach to multi-agent milps: finite-time feasibility and performance guarantees. Automatica **103**, 141–150 (2019)

[10] Mosteo, A.R., Montano, L.: Simulated annealing for multi-robot hierarchical task allocation with flexible constraints and objective functions. In: Workshop on Network Robot Systems: Toward Intelligent Robotic Systems Integrated with Environments. Int. Conf. on Intelligent Robots and Systems (2006). Citeseer

[11] Gerkey, B.P., Mataric, M.J.: Sold!: Auction methods for multirobot coordination. IEEE transactions on robotics and automation **18**(5), 758–768 (2002)

[12] Choi, H.-L., Brunet, L., How, J.P.: Consensus-based decentralized auctions for robust task allocation. IEEE transactions on robotics **25**(4), 912–926 (2009)

[13] Chen, S., Lin, T.X., Al-Abri, S., Arkin, R.C., Zhang, F.: Hybrid susd-based task allocation for heterogeneous multi-robot teams. In: 2023 IEEE International Conference on Robotics and Automation (ICRA), pp. 1400–1406 (2023). https://doi.org/10.1109/ICRA48891.2023.10161349

[14] Al-Abri, S., Lin, T.X., Nelson, R.S., Zhang, F.: A derivative-free distributed optimization algorithm with applications in multi-agent target tracking. In: 2021 American Control Conference (ACC), pp. 3844–3849 (2021). https://doi.org/10.23919/ACC50511.2021.9483125

[15] Yang, R., Zhang, F., Hou, M.: Oceanplan: Hierarchical planning and replanning for natural language auv piloting in large-scale unexplored ocean environments. In: Proceedings of the 18th International Conference on Underwater Networks & Systems. WUWNET '24. Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3699432.3699496 . https://doi.org/10.1145/3699432.3699496

[16] Wu, Y., Tao, Y., Li, P., Shi, G., Sukhatmem, G.S., Kumar, V., Zhou, L.: Hierarchical LLMs In-the-loop Optimization for Real-time Multi-Robot Target Tracking under Unknown Hazards (2024). https://arxiv.org/abs/2409.12274

[17] Obata, K., Aoki, T., Horii, T., Taniguchi, T., Nagai, T.: LiP-LLM: Integrating Linear Programming and dependency graph with Large Language Models for multi-robot task planning (2024). https://arxiv.org/abs/2410.21040

[18] Kannan, S.S., Venkatesh, V.L., Min, B.-C.: Smart-llm: Smart multi-agent robot task planning using large language models. arXiv preprint arXiv:2309.10062 (2023)

[19] Prieto, S., Soto, B.: Large language models for robot task allocation. (2024). https://doi.org/10.22260/ICRA2024/0007

[20] Li, Q., Zhang, L., Mak-Hau, V.: Synthesizing mixed-integer linear programming models from natural language descriptions. (2023). https://api.semanticscholar.org/CorpusID:265456244

[21] AhmadiTeshnizi, A., Gao, W., Brunborg, H., Talaei, S., Lawless, C., Udell, M.: OptiMUS-0.3: Using Large Language Models to Model and Solve Optimization Problems at Scale (2025). https://arxiv.org/abs/2407.19633

[22] Jiang, C., Shu, X., Qian, H., Lu, X., Zhou, J., Zhou, A., Yu, Y.: LLMOPT: Learning to Define and Solve General Optimization Problems from Scratch (2025). https://arxiv.org/abs/2410.13213

[23] Lawless, C., Li, Y., Wikum, A., Udell, M., Vitercik, E.: LLMs for Cold-Start Cutting Plane Separator Configuration (2024). https://arxiv.org/abs/2412.12038

[24] Wang, T., Yu, W.-Y., She, R., Yang, W., Chen, T., Zhang, J.: Leveraging Large Language Models for Solving Rare MIP Challenges (2024). https://arxiv.org/abs/2409.04464

[25] Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q.V., Zhou, D., Chen, X.: Large language models as optimizers. ArXiv **abs/2309.03409** (2023)

[26] Hoffman, K.L., Padberg, M., Rinaldi, G., *et al.*: Traveling salesman problem. Encyclopedia of operations research and management science **1**, 1573–1578 (2013)

[27] Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M.P., Dupont, E., Ruiz, F.J., Ellenberg, J.S., Wang, P., Fawzi, O., *et al.*: Mathematical discoveries from program search with large language models. Nature **625**(7995), 468–475 (2024)

[28] Bateman, M., Katz, N.: New bounds on cap sets. Journal of the American Mathematical Society **25**(2), 585–613 (2012)

[29] Liu, F., Lin, X., Wang, Z., Yao, S., Tong, X., Yuan, M., Zhang, Q.: Large language model for multi-objective evolutionary optimization. arXiv preprint

arXiv:2310.12541 (2023)

[30] Liu, S., Chen, C., Qu, X., Tang, K., Ong, Y.-S.: Large language models as evolutionary optimizers. In: 2024 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2024). https://doi.org/10.1109/CEC60901.2024.10611913

[31] Lange, R., Tian, Y., Tang, Y.: Large language models as evolution strategies. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 579–582 (2024)

[32] Cai, J., Xu, J., Li, J., Yamauchi, T., Iba, H., Tei, K.: Exploring the improvement of evolutionary computation via large language models. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 83–84 (2024)