Analytical Modeling of Task Allocation for Distributed Anthropomorphic Robots in Mission-Critical Environments

Zhishang Wang, Yassine Khedher, Khanh N. Dang, and Abderazek Ben Abdallah
School of Computer Science and Engineering
University of Aizu
Aizu-Wakamatsu, Fukushima; Japan
{zwang, m5281019, khanh, benab}@u-aizu.ac.jp

***Abstract*—Efficient task allocation among mobile agents is critical in high-stakes scenarios such as disaster response, battlefield coordination, and emergency healthcare. This paper introduces a framework for task allocation across distributed anthropomorphic robots (androids) operating in dynamic, resource-constrained environments. The system leverages decentralized decision-making, context-aware prioritization, and adaptive communication protocols to maintain robust performance under intermittent connectivity, limited energy availability, and unpredictable task demands. We present an analytical model that formalizes the interplay between agent autonomy, resource limitations, and task urgency, enabling adaptive behavior in volatile conditions. A preliminary evaluation indicates feasibility and effectiveness of the proposed approach, with notable gains in task distribution efficiency, latency reduction, and fault tolerance compared to centralized alternatives. In particular, the proposed leader-based allocation method that incorporates force, energy, and timing constraints achieves about 2.0× higher success rate than existing approaches with 30 tasks and 100 androids, and more than 3.5× higher success rate with 50 tasks and 300 androids. These results lay a foundation for scalable, resilient coordination mechanisms in mission-critical mobile networks.**

***Index Terms*—Android Systems, Distributed Task Allocation, Analytical Modeling, Mission-Critical**

I. Introduction

Labor shortages have become a critical global challenge. In 2025, nearly three-quarters of employers worldwide reported difficulty filling job vacancies, which is almost double the proportion from a decade earlier [1]. This demographic trend has led to increased reliance on androids and autonomous robotic systems to supplement human labor, particularly in hazardous, physically demanding, or logistically complex environments.

Robots and androids are increasingly deployed in diverse domains. In defense and disaster response, remotely operated systems are used for demining and inspecting hazardous sites [2, 3]. In logistics, large-scale operations such as Amazon's fulfillment centers employ more than 750,000 heterogeneous robots, including mobile drive units, robotic arms, and packaging systems that coordinate to transport, sort, and prepare goods [4]. Similar swarm-based solutions appear in warehouse automation [5] and search-and-rescue exploration [6]. These examples highlight the growing importance of coordinated multi-robot systems across industrial, commercial, and emergency applications.

Despite such progress, many deployments remain small in scale, focused on homogeneous teams, or constrained by centralized control. Even heterogeneous systems, such as Amazon's fulfillment network, depend upon predefined roles and centralized scheduling, leaving open questions about scalability and adaptability in complex, real-world scenarios.

At the research level, task allocation has been studied through centralized optimization [7, 8], language-model-based planning [9, 10], modular skill libraries [11], and decentralized coordination with safety or resilience guarantees [12, 13]. These approaches have advanced the field, yet most assume one-to-one task mappings, small team sizes, or fixed role definitions. Few explicitly incorporate dynamic factors such as spatial proximity, force capacity, or energy availability, which are essential for large-scale heterogeneous android networks.

To overcome the above challenges, we propose a leader-based task assignment method in distributed autonomous android systems, where a leader android allocates tasks when many androids are in different locations, with different statuses and capability types, aiming for effective and efficient assignment. The main contributions are as follows:

- We present a distributed system for heterogeneous autonomous androids, where tasks with diverse spatiotemporal and physical requirements arrive sequentially, and idle androids collaboratively maintain task and status information.
- We propose an analytical modeling of distributed task allocation algorithm for android systems in mission-critical environments. Building on this modeling, we describe an algorithm that ensures feasible assignment by jointly considering android types, force capabilities, battery constraints, and timing requirements, while handling retry and expiration conditions.
- We introduce a lightweight leader selection and substitution scheme, where androids deterministically select a leader based on compute capacity and monitor its liveness through heartbeat signals. This mechanism provides robustness against failures and maintains continuous de-

centralized coordination.
- A physical android prototype was developed to support function integration under real-world conditions.

II. Related Work

Recent research on multi-agent systems has explored a wide range of strategies to enable collaboration, adaptability, and robustness across heterogeneous agents in dynamic environments. Among them, a growing body of work leverages large language models (LLMs) for task planning and coordination. COHERENT [7], SMART-LLM [10], CoELA [8], and IRoT [9] demonstrate LLM-based frameworks that support flexible task decomposition, adaptive policy generation, and decentralized cooperation across diverse robot types. Vision-language models have also been integrated into multi-robot navigation, as in Co-NavGPT [14], which enables map fusion and coordinated frontier assignment through semantic reasoning. Practical applications have motivated heterogeneous assistive robot designs for home environments [15], while broader surveys such as [11] have emphasized key challenges in multi-robot search and rescue, including active perception and simulation-to-reality transfer. Other lines of work address operational resilience and coordination under uncertainty, including reallocation under environmental perturbations [12], dynamic coalition formation [16], and distributed safe navigation with connectivity guarantees [13]. However, many of these approaches still assume persistent global state awareness, task-specific training, or frequent inter-agent communication, which can be difficult to maintain in real-world, uncertain, or bandwidth-limited settings. These constraints hinder adaptability during execution and limit robustness of multi-agent cooperation in practical deployments. In contrast to previous approaches that assume stable conditions or fixed agent-task mappings, our work addresses task allocation in environments with dynamic task arrivals, heterogeneous capabilities, and limited communication. We introduce a decentralized framework in which androids make assignment decisions through local coordination, physical constraint filtering, and lightweight leader-based selection to maintain scalability and robustness without requiring global state.

III. Leader-Based Assignment of Tasks in Distributed Autonomous Android Systems

We propose a leader-based task assignment method for distributed autonomous android system, as illustrated in Figure 2. In this setting, a stream of heterogeneous tasks arrives over time. Each task has a specific location and duration, and may optionally include constraints on when it must start or finish. Each task requires one or more types of androids, with both per-android and total force requirements.

Each android in the system belongs to a specific type and is located at a known position. Additional attributes include its remaining battery time, task-dependent battery consumption rate, and available force capacity. In this work, we only consider idle androids; those that are busy or in a fault state are excluded from the assignment process.

Now, given a sequence of tasks and a group of androids, we process the tasks sequentially in a first-in-first-out (FIFO) manner. For each task, the system searches for a group of idle androids that satisfy the required types, quantities, force conditions, and timing constraints. Spatial proximity is also considered to enable faster response. If a valid set is found, the task is assigned and considered fulfilled. Otherwise, the task is temporarily deferred or discarded, depending on retry conditions and expiration status, which are defined in the detailed algorithm later.

The following subsections describe the formal denotation of tasks and androids, followed by the proposed assignment algorithm.

A. Notation for Androids and Tasks

Let $\mathcal{A} = a_1, a_2, \ldots, a_n$ denote the set of N androids. Each android a_i is represented by the tuple:

$a_i = (i, p_i, b_i, \beta_i, \gamma_i, f_{i,\gamma}, c_i)$

- i: android ID
- p_i: current 2D location
- b_i: remaining battery time [min]
- β_i: task-dependent battery consumption rate
- β_i^{move}: battery consumption rate while moving (per step)
- $\gamma_i \in \Gamma$: android type (where Γ is the set of all types)
- $f_{i,\gamma}$: force level for type γ (e.g., maximum exertable force [N])
- c_i: compute capacity, reflecting android's available processing capability

At any given time, a designated idle android is assigned the role of leader, denoted as a^{leader}. The leader is responsible for coordinating task assignment, querying android information, maintaining spatial indices, and retrying unfulfilled tasks.

Let $\mathcal{T}^{\text{unassigned}} = \text{task}_1, \text{task}_2, \ldots, \text{task}_m$ be the set of M unassigned tasks. A task_j is defined by the tuple: $\text{task}_j = (l_j, w_j, t_j^{\text{req}}, \Gamma_j^{\text{req}}, \mathcal{F}_j, t_j^{\text{start}}, t_j^{\text{start\_tol}}, t_j^{\text{finish}}, t_j^{\text{finish\_tol}}, r_j)$

- j: task ID
- l_j: task location
- w_j: duration
- t_j^{req}: request time
- $\Gamma_j^{\text{req}} = (\gamma, n_{j,\gamma}^{\text{req}})$: required android types and number
- $\mathcal{F}_j = (\gamma, f_{j,\gamma}^{\min})$: per-agent force requirement
- $t_j^{\text{start}}, t_j^{\text{start\_tol}}$: task start time and tolerance
- $t_j^{\text{finish}}, t_j^{\text{finish\_tol}}$: task finish time and tolerance
- r_j: remaining retry count

B. Leader Selection and Substitution

At the beginning of the task allocation process, one android is designated as the leader, denoted by a^{leader}. The leader is selected as the android with the highest compute capacity c_i. In case of a tie, the android with the smallest index is chosen. This leader is responsible for collecting the status of idle androids and unassigned tasks, performing assignment decisions, and notifying the selected androids.

Each android a_i monitors periodic heartbeat signals sent by the leader at a fixed interval τ. a_i records the time t_i^{last} when

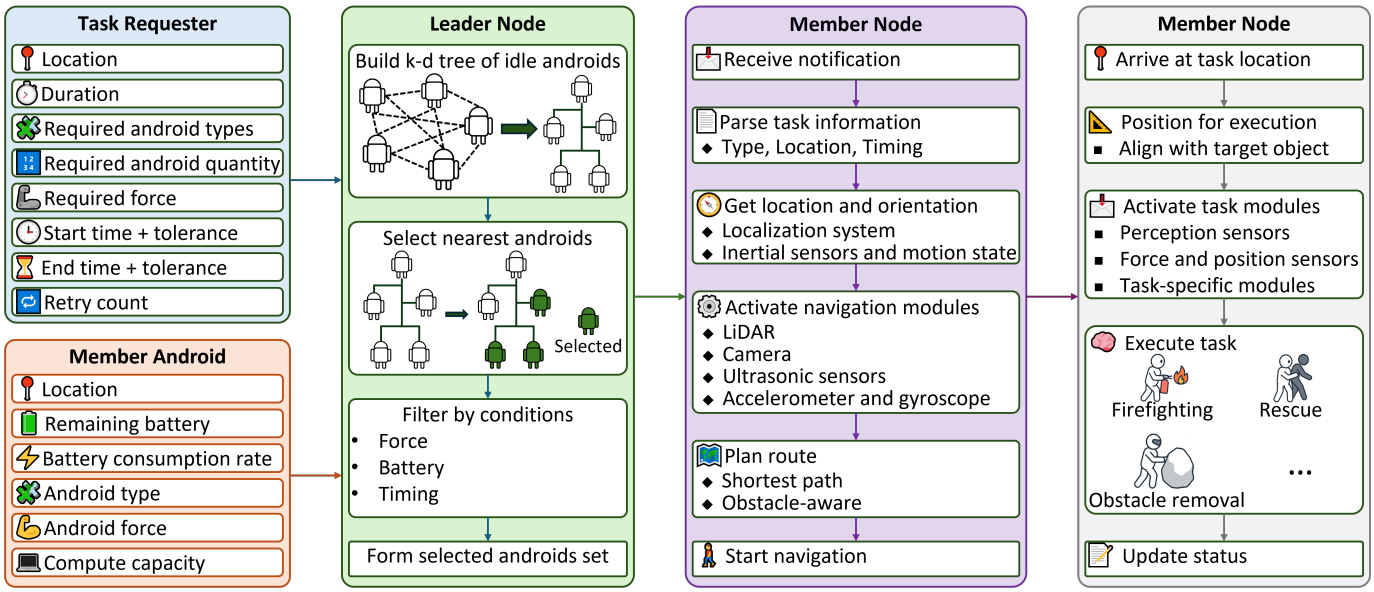


Fig. 1: Overview of the proposed distributed android coordination framework for critical missions. Task requesters specify requirements (e.g., location, type, timing, force), while android nodes report their status (e.g., location, battery, capability). The leader node builds a k-d tree to select nearby candidates, filters them based on constraints, and assigns tasks. Selected member nodes parse instructions, localize themselves, activate navigation modules, and autonomously reach the task site. Upon arrival, they align with the target, activate task-related modules (sensing and actuation), execute missions such as firefighting, rescue, or obstacle removal, and update status to complete the task cycle.

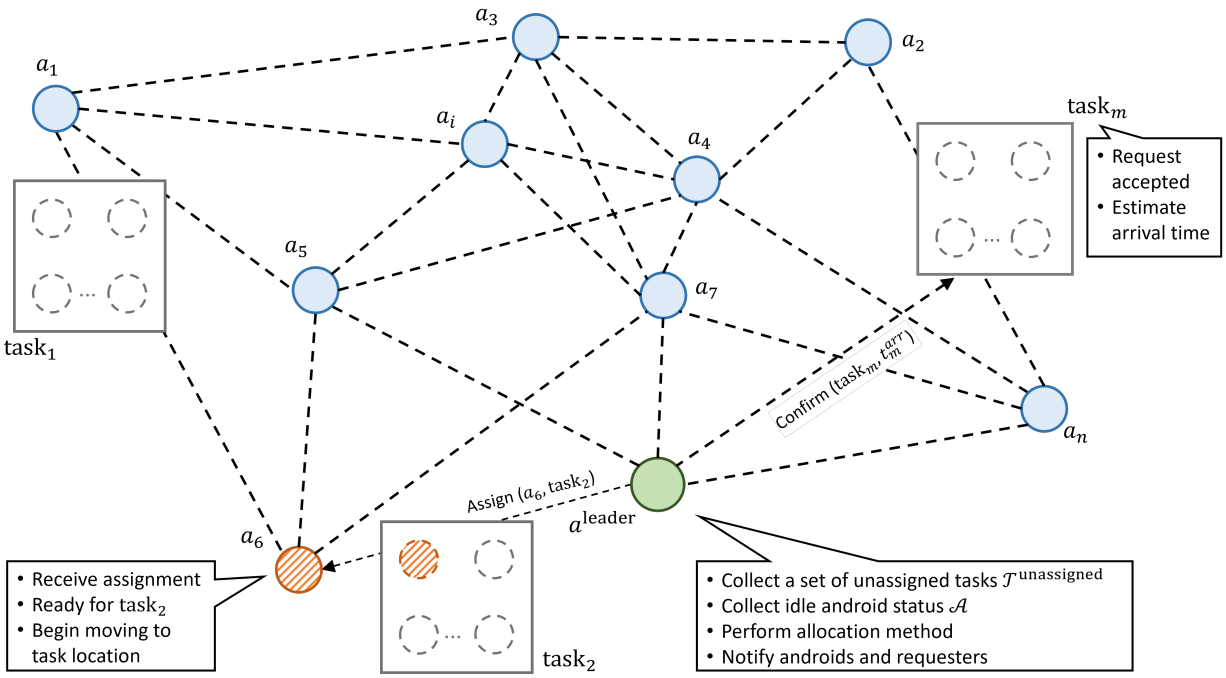


Fig. 2: Overview of the leader-based task assignment method for distributed autonomous android system. Each android is represented as a circle for clarity. Several representative task locations are shown (task_1, task_2, task_m). Dashed-line circles at each task location indicate where androids are still needed to satisfy task requirements. Blue circles indicate idle androids. The green circle represents the leader android, which collects the status of idle androids and the list of unassigned tasks, performs assignment based on proximity, type, force, and timing constraints, and notifies the selected androids and the task requesters. The orange circle represents an assigned android currently heading toward its task. The system operates through decentralized communication and supports dynamic coordination across spatially distributed agents.

Algorithm 1 BuildKDTree(P, d)

Require: Set of 2D points $P \subseteq \mathbb{R}^2$, recursion depth d
Ensure: Root node of k-d tree
Recursively call Steps 1 to 6:
1: **If** $P = \emptyset$, **return** null
2: Select splitting axis: axis $= d \bmod 2$
If axis $= 0$: sort P in ascending order of x_i;
If axis $= 1$: sort P in ascending order of y_i;
3: Let m be the index of the median:
$$m = \left\lfloor \frac{|P|}{2} \right\rfloor$$
4: Let p^* be the median point:
$$p^* = P[m]$$
5: Recursively build left and right subtrees:
left $\leftarrow$ BuildKDTree$(P[< m],\ (d+1) \bmod 2)$
right $\leftarrow$ BuildKDTree$(P[> m],\ (d+1) \bmod 2)$
6: **Return** a node:
Node$(p^*, \text{left}, \text{right}, \text{axis} = d)$

it last received a heartbeat. If the elapsed time since the last heartbeat exceeds a pre-defined time threshold,

$$t^{\text{now}} - t_i^{\text{last}} > k \cdot \tau, \tag{1}$$

where $k > 1$ is a predefined timeout factor, the android concludes that the leader has failed. This decision is made independently by each android. Since the subsequent leader substitution procedure is deterministic and consistent across nodes, a new leader will be agreed upon naturally without requiring explicit coordination.

After an android detects the failure of the current leader, the following steps are executed to elect a new leader in a distributed manner.

- **Step 1: Self-Nomination**
  Each idle android that detects leader failure broadcasts a nomination message of the form: NOMINATE(a_i, c_i), where a_i is its own ID and c_i is its compute capacity.
- **Step 2: Nomination Message Reception**
  Each android receives NOMINATE messages broadcast by others. It temporarily stores the received nominations for a short election period of duration Δt^{select}.
- **Step 3: Deterministic Selection**
  After the election window ends, each android locally selects the new leader $a^{\text{leader'}}$ with highest compute capacity.
- **Step 4: Leader Announcement**
  The selected android $a^{\text{leader'}}$ broadcasts a confirmation message: LEADER_ANNOUNCE$(a^{\text{leader'}})$, to ensure all members explicitly acknowledge the new leader.

C. Task Allocation Algorithm in Distributed Android System

A complete workflow of the proposed task allocation algorithm is illustrated in Figure 3. The process begins with the leader node constructing a k-d tree to index the spatial proximity of all idle androids. Details of the k-d tree construction procedure are provided in Algorithm 1. Once the tree is built, the task allocation proceeds through the following six steps.

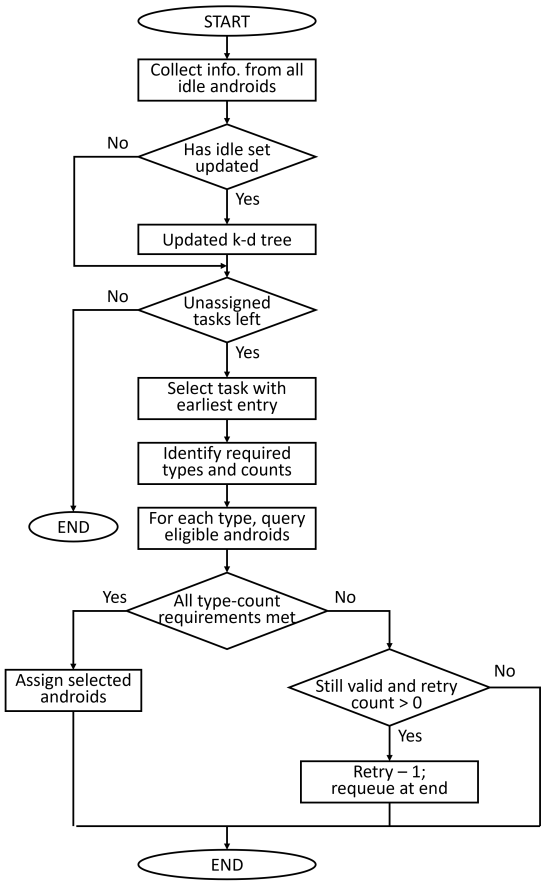


Fig. 3: Flowchart of task allocation in distributed android system. The leader periodically collects status information from all idle androids and determines if the idle set has changed. If so, it updates the spatial index (k-d tree) used for proximity-based selection. When unassigned tasks exist, the leader selects the task with the earliest entry and extracts the required android types and counts. For each type, it queries the set of eligible idle androids. If all type-count constraints are satisfied, the task is assigned. If not, the task is either requeued (with decreased retry count) or discarded, depending on its validity and remaining retry count.

- **Step 1: Collect Android Status and Heartbeat Validation**
  At the core of the task allocation process is a continuous loop operated by the leader android. In each iteration, the leader first updates its knowledge of the system state by collecting status information from all idle androids. This includes each android's type, current location, remaining battery level, and available force.
  Androids that have just completed a task, finished charging, or moved to a new location immediately transmit their new status to the leader. To ensure fault tolerance, each idle android is also required to periodically send a heartbeat signal. If the leader does not receive a heartbeat from a previously idle android within a predefined time-

out window, it removes that android from the idle set to avoid selecting failed or unreachable units.

- **Step 2: Rebuild Spatial Indices (k-d Trees)**
  If the idle set has changed since the last allocation cycle (e.g., due to arrival, disappearance, or update of any android), the leader rebuilds the spatial indices. Specifically, for each android type γ, a separate k-d tree KD_γ is constructed to index the 2D spatial positions of all currently idle androids of that type, as shown in Algorithm 1: $KD_\gamma = \text{BuildKDTree}(P_\gamma, 0)$, where P_γ is the list of positions of idle androids of type γ. These spatial indices enable efficient nearest-neighbor search in the subsequent selection steps.
- **Step 3: Check for Unassigned Tasks and Select Next Task**
  The leader then checks whether any unassigned tasks remain. If the unassigned set $\mathcal{T}^{\text{unassigned}}$ is empty, the system continues monitoring until new tasks arrive or androids become available.
  Otherwise, the leader selects the next task task_j from the queue, using the earliest request time as priority: $\text{task}_j = \arg\min_{\text{task}_k \in \mathcal{T}^{\text{unassigned}}} t_k^{\text{req}}$ This ensures that tasks are processed in temporal order, respecting their urgency or submission time.
- **Step 4: Retrieve Candidate Androids via k-d Tree Queries**
  Once the task task_j is selected, the leader must find a suitable set of androids to fulfill the task's requirements. Each task may require multiple androids of different types, specified by the set Γ_j^{req}, where each type γ has an associated number $n_{j,\gamma}^{\text{req}}$ of required androids.
  To efficiently find androids that are spatially close to the task location, the leader uses the previously constructed k-d trees KD_γ, which index the positions of idle androids for each type γ. For each $\gamma \in \Gamma_j^{\text{req}}$, the leader performs a nearest-neighbor query on the corresponding k-d tree to obtain a set of candidate androids:

$$B_{j,\gamma} = \min(K \cdot n_{j,\gamma}^{\text{req}},\ |\mathcal{A}_{\text{idle},\gamma}|)$$

$$\tilde{\mathcal{A}}_{j,\gamma} = \text{Query}(KD_\gamma, \text{location}_j, B_{j,\gamma})$$

  , where $\tilde{\mathcal{A}}_{j,\gamma}$ is the retrieved candidate set for type γ, and location_j denotes the spatial coordinates of the task.
  The parameter $K > 1$ is an oversampling factor used to increase robustness. It ensures that even after filtering (e.g., due to insufficient force, low battery, or time violations), a sufficient number of eligible androids might still be available. This is crucial for increasing the task assignment success rate, especially under high-load or sparsely distributed android scenarios.
  Note that the retrieved candidate sets are provisional: at this stage, no feasibility checks beyond type and spatial proximity are applied. Further filtering is performed in the next step.
- **Step 5: Eligibility Filtering**
  For each candidate set $\tilde{\mathcal{A}}_{j,\gamma}$ retrieved in Step 4, the leader performs eligibility filtering to ensure that androids satisfy all necessary constraints. Each android $a_i \in \tilde{\mathcal{A}}_{j,\gamma}$ is evaluated according to the following conditions:
  - **Force Constraint:** $f_{i,\gamma} \geq f_{j,\gamma}^{\min}$. The android must have sufficient force to meet the minimum required for type γ.
  - **Battery Constraint:** $b_i^{\text{remain}} \geq b_i^{\text{travel}} + b_i^{\text{task}}$, where b_i^{travel} is the estimated battery needed to reach the task location and b_i^{task} is the consumption for executing the task.
  - **Timing Constraint:** $t_j^{\text{start}} - \delta_j^{\text{start}} \leq t_i^{\text{arrive}} \leq t_j^{\text{finish}} + \delta_j^{\text{finish}}$, where t_i^{arrive} is computed from the android's current available time and estimated travel duration. The android must arrive within the task's tolerance window.

  Only androids satisfying *all three* constraints are retained in the filtered set $\hat{\mathcal{A}}_{j,\gamma}$. This process is repeated independently for all required types $\gamma \in \Gamma_j^{\text{req}}$. If, for any type γ, the number of eligible androids is less than the required count $n_{j,\gamma}^{\text{req}}$, then the task is considered infeasible under current conditions, and fallback handling is triggered in the next step.
- **Step 6: Incomplete Assignment Handling**
  If fewer than $n_{j,\gamma}^{\text{req}}$ androids of any required type γ pass the eligibility filters, the system initiates the following resolution process:
  - **Deadline Violation:** If the current time exceeds the allowable time window (start or finish constraints), the task is discarded.
  - **Retry Available:** If the retry count $r_j > 0$, it is decremented and the task is reinserted at the end of the unassigned task queue.
  - **No Retry Left:** If no retry attempts remain, the task is permanently discarded without assignment.

IV. Evaluation

A. Evaluation Methodology

To evaluate the performance of the proposed method, the key performance metric used is the **success rate**, which is defined as the ratio of tasks successfully assigned to androids to the total number of tasks. A task is considered successfully assigned only if all its required androids are allocated in a way that satisfies capability type, minimum force, sufficient battery, and, optionally, timing constraints such as start and end windows with tolerances.

The task list and the android list are generated according to configurable parameters detailed in Tables I and II. Each task may require multiple androids of specific types, and may include temporal constraints with tolerances. Androids are heterogeneous in terms of capability type, battery capacity, force, and availability. The retry count for each task is set to 1, meaning the allocation for each task can be attempted one more time.

We test various combinations of task and android counts. Specifically, we vary the number of tasks (e.g., 10, 20, 30,

and 50) and the number of androids (e.g., 50, 100, 150, 200, 250, and 300). For each combination, we run 100 independent simulations. In every iteration, a new task list and android list are generated to simulate real-world variation, and results are averaged over 100 runs to obtain reliable estimates of performance.

The proposed method is compared with two baseline methods. The work IRoT [9] considers capability type, minimum force, and battery constraints, but does not check time feasibility during candidate selection. The method COHERENT [7], in contrast, only filters candidates by capability type during selection.

In addition to simulation-based evaluation, we built a physical android prototype to demonstrate embodiment of our system design. The prototype consists of 3D-printed joints and limb structures assembled onto a torso frame, which is designed to accommodate future integration of modular actuators and embedded computing units (e.g., FPGA). All structural components were fabricated using a Bambu Lab P1S Combo 3D printer.

B. Evaluation Results

We evaluate the proposed method from two perspectives. First, we analyze how the success rate changes as the number of androids increases under different task workloads. Second, we examine how the success rate changes as the number of tasks increases while fixing the android group size.

Figure 4 presents the success rate as the number of androids increases under workloads of 10, 20, 30, and 50 tasks. The results show that the proposed method consistently outperforms IRoT and COHERENT across all workloads, especially when the number of androids is limited. This indicates that our approach achieves better resource efficiency under constrained conditions. As the number of androids grows, the success rate of all methods increases, but the proposed method sustains its advantage, demonstrating improved scalability.

Figure 5 illustrates the success rate as the number of tasks increases, under fixed android team sizes ranging from 50 to 300. As expected, all methods experience declining success rates with increasing workloads. However, the proposed method consistently sustains significantly higher success rates compared to IRoT and COHERENT, particularly under heavy loads. This robustness under stress confirms the advantage of our approach in balancing resource constraints with demand growth. Together, these results validate that the proposed method achieves superior task allocation performance across a wide range of conditions.

To demonstrate the feasibility of real-world deployment, we present physical components related to the proposed android framework. The prototype android (Fig. 6a) features 3D-printed limb structures mounted on a torso frame, designed to accommodate future integration of actuators and embedded computing. Fig. 6b shows AIzuHand, a neuromorphic prosthetic hand with EMG-based control and real-time feedback, which may serve as a potential manipulation module in future extensions of our android [17].

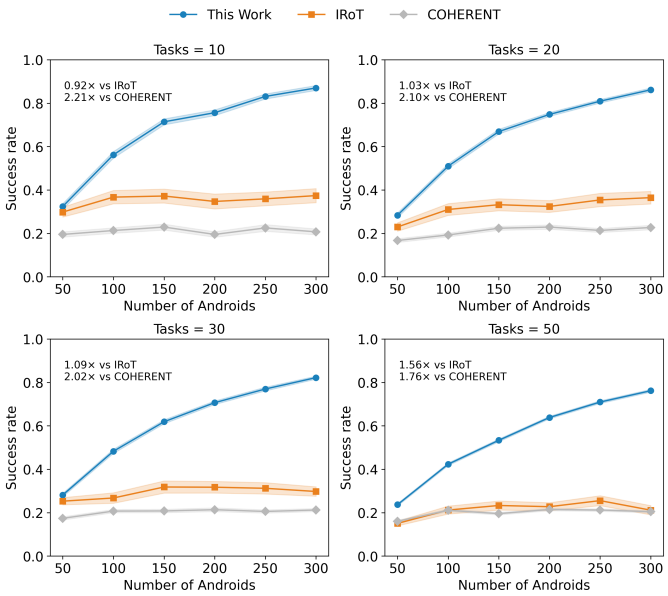


Fig. 4: Evaluation of the proposed method in terms of task-fulfillment success rate compared to IRoT [9] and COHERENT [7]. The plots show the success rate as a function of the number of available androids, under workloads of 10, 20, 30, and 50 tasks. Each subplot corresponds to a fixed number of tasks, while the curves represent different methods. The proposed method consistently achieves higher success rates, particularly when the number of androids is limited.

V. Discussion and Conclusion

This paper proposes a leader-based task allocation method for heterogeneous android networks. The approach selects androids not only by proximity but also through realistic feasibility checks, including force requirements, battery consumption for both movement and execution, and strict as well as tolerant timing constraints. By integrating these practical factors, the method avoids infeasible allocations and ensures higher rates of complete task fulfillment. To support real-world applicability, we also developed a partial android prototype focused on structural integration, including preliminary work toward incorporating a prosthetic hand for object manipulation.

We analyzed the results from two complementary perspectives: varying the number of androids under fixed task loads, and varying the number of tasks under fixed android team sizes. From the android-scaling perspective, our method consistently achieved higher success rates than IRoT and COHERENT. For example, with 30 tasks and 100 androids, the proposed method reached a mean success rate of 0.48, compared to 0.27 for IRoT and 0.21 for COHERENT. This represents a nearly two-fold improvement in resource-constrained conditions. Importantly, while the baselines plateaued and failed to benefit from additional androids, our method continued to scale upward as more resources became available.

From the task-scaling perspective, the proposed method

TABLE I: Configuration of Task Parameters Based on Spatiotemporal and Physical Requirements

Parameter	Notation	Description	Value / Range
TaskID	j	Unique ID per task	1 to Number of Tasks
Location	l_j	Task 2D coordinates	(0, 0) to (99, 99)
Duration	w_j	Required task execution time	10 to 60 min
StartTime	t_j^{start}	Start time constraint	50% of tasks: 1$^{\text{st}}$ to 360$^{\text{th}}$ minute
StartTimeTolerance	$t_j^{\text{start\_tol}}$	Tolerance for delayed start	5 to 30 min
FinishTime	t_j^{finish}	End time constraint	50% of tasks: 11$^{\text{th}}$ to 420$^{\text{th}}$ minute
FinishTimeTolerance	$t_j^{\text{finish\_tol}}$	Tolerance for delayed finish	5 to 30 min
RetryCount	r_j	Max retry attempts for task	1
RequiredType	γ	Required android type	['A', 'B', 'C']
RequiredNum	$n_{j,\gamma}^{\text{req}}$	Androids required for this type	1 – 3
MinForcePerAndroid	$\gamma, f_{j,\gamma}^{\min}$	Min required force	1 – 10 Newton

TABLE II: Configuration of Android Parameters Based on Type, Force, and Battery Characteristics

Parameter	Notation	Description	Value / Range
AndroidID	i	Unique ID per android	1 to Number of Androids
Location	p_i	Android 2D coordinates	(0, 0) to (99, 99)
Type	$\gamma_i \in \Gamma$	Android type / capability	['A', 'B', 'C']
Force	$f_{i,\gamma}$	Physical strength	1 – 10 N
RemainingBattery	b_i	Initial battery status (in minutes)	10 to 120 min
TaskConsumptionRate	β_i	Battery consumption rate to perform task	0.8 for A, 1.0 for B, 1.2 for C
StepConsumptionRate	β_i^{move}	Battery consumption per step	5% of full capacity

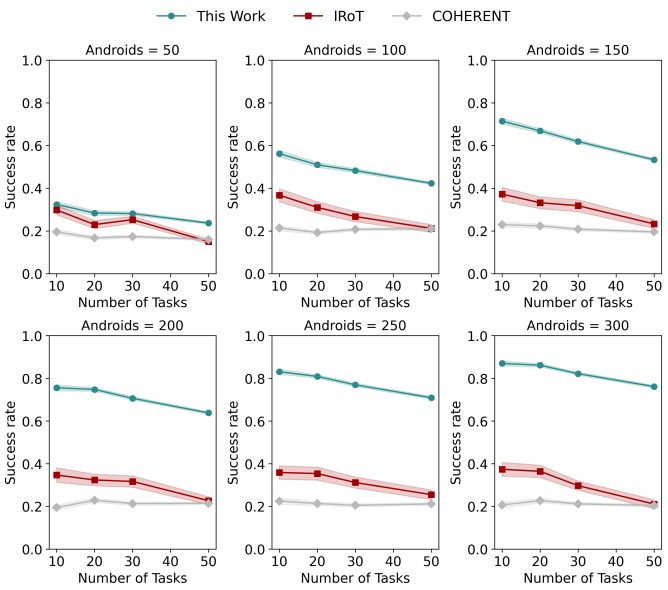
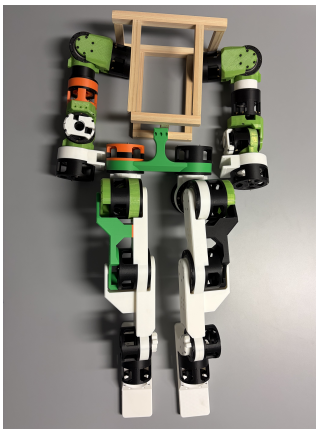


Fig. 5: Evaluation of the proposed method in terms of task-fulfillment success rate compared to IRoT [9] and COHERENT [7]. The plots show the success rate as a function of the number of tasks, under android team sizes of 50, 100, 150, 200, 250, and 300. Each subplot corresponds to a fixed number of androids, while the curves represent different methods. The proposed method maintains higher success rates as workload increases, demonstrating enhanced robustness under heavy task demands.



(a) The developed android prototype. The prototype consists of articulated limb structures assembled from 3D-printed parts using modular joints. A wooden frame is temporarily used as the torso base to support initial mechanical integration and future component embedding.

(b) AIzuHand: AIzuHand is a 3D-printed, low-cost prosthetic hand designed for EMG-based control with real-time feedback, a neuromorphic prosthetic hand with sensory-motor integration. This figure is revised from the work [17].

Fig. 6: Prototype android (a) and AIzuHand prosthetic hand (b) as physical components toward real-world deployment.

also showed stronger robustness under increasing workloads. With 50 tasks and 300 androids, our method achieved a success rate of 0.76, while IRoT and COHERENT reached 0.21 and 0.20, respectively, which corresponds to more than 3.5 times improvement. This demonstrates that the allocation strategy sustains performance even under heavy demands, where competing methods collapse in success rate. These findings highlight that the performance gain comes from mod-

eling real-life constraints that prior frameworks overlook. By explicitly incorporating energy, force, and temporal feasibility, the proposed method can maintain resilience across a wider range of conditions. Nevertheless, limitations remain. The current version does not yet address fault reactions, such as androids failing during task execution, and some parameters were simplified for validation. Future work will refine these aspects and extend evaluation to physical android networks.

In conclusion, the proposed leader-based allocation method offers a practical and scalable strategy for heterogeneous android teamwork. By coupling realistic constraints with leader-driven coordination, it achieves consistent improvements of up to nearly four times better than existing approaches while maintaining scalability under increasing workloads.

ACKNOWLEDGMENT

This work was supported by The University of Aizu, CRF-P-37-2025.

REFERENCES

[1] *The Latest Labor Shortage Trends & Statistics (2025)*. Exploding Topics. [Online]. Available: https://explodingtopics.com/blog/labor-shortage-stats. Accessed: Oct. 14, 2025.

[2] *Safer Demining through Technology*. Government of Japan. [Online]. Available: https://www.japan.go.jp/kizuna/2025/03/safer_demining_through_technology.html. Accessed: Oct. 14, 2025.

[3] *Application of Robot Technology*. Tokyo Electric Power Company Holdings, Inc. [Online]. Available: https://www.tepco.co.jp/en/decommision/principles/robot/index-e.html. Accessed: Oct. 14, 2025.

[4] *Amazon has deployed more than 750,000 robots across its operations*. Amazon Robotics. [Online]. Available: https://www.aboutamazon.com/news/operations/amazon-robotics-robots-fulfillment-center. Accessed: Oct. 14, 2025.

[5] *Automating order picking warehouses with a swarm*. SSI Schaefer. [Online]. Available: https://www.materialfluss.de/agvs-and-robotics/automating-order-picking-warehouses-with-a-swarm.htm. Accessed: Oct. 14, 2025.

[6] *Navigating the Swarm*. Osaka University. [Online]. Available: https://resou.osaka-u.ac.jp/en/research/2025/20250106_1. Accessed: Oct. 14, 2025.

[7] Kehui Liu et al. "Coherent: Collaboration of heterogeneous multi-robot system with large language models". In: *arXiv preprint arXiv:2409.15146* (2024). DOI: 10.48550/arXiv.2409.15146.

[8] Hongxin Zhang et al. "Building cooperative embodied agents modularly with large language models". In: *arXiv preprint arXiv:2307.02485* (2023). DOI: 10.48550/arXiv.2307.02485.

[9] Xinzhu Liu et al. "Leveraging large language model for heterogeneous ad hoc teamwork collaboration". In: *arXiv preprint arXiv:2406.12224* (2024). DOI: 10.48550/arXiv.2406.12224.

[10] Shyam Sundar Kannan, Vishnunandan LN Venkatesh, and Byung-Cheol 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. 12140–12147. DOI: 10.1109/IROS58592.2024.10802322.

[11] Jorge Pena Queralta et al. "Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision". In: *IEEE Access* 8 (2020), pp. 191617–191643. DOI: 10.1109/ACCESS.2020.3030190.

[12] Siddharth Mayya et al. "Resilient Task Allocation in Heterogeneous Multi-Robot Systems". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 1327–1334. DOI: 10.1109/LRA.2021.3057559.

[13] Andrea Miele, Martina Lippi, and Andrea Gasparri. "A Distributed Framework for Integrated Task Allocation and Safe Coordination in Networked Multi-Robot Systems". In: *IEEE Transactions on Automation Science and Engineering* 22 (2025), pp. 11219–11238. DOI: 10.1109/TASE.2025.3532023.

[14] Bangguo Yu, Hamidreza Kasaei, and Ming Cao. "Co-navgpt: Multi-robot cooperative visual semantic navigation using large language models". In: *arXiv preprint arXiv:2310.07937* (2023). DOI: 10.48550/arXiv.2310.07937.

[15] Patrick Benavidez et al. "Design of a home multi-robot system for the elderly and disabled". In: *2015 10th System of Systems Engineering Conference (SoSE)*. IEEE, pp. 392–397. DOI: 10.1109/SYSOSE.2015.7151907.

[16] Lucas C. D. Bezerra, Ataíde M. G. dos Santos, and Shinkyu Park. "Learning Policies for Dynamic Coalition Formation in Multi-Robot Task Allocation". In: *IEEE Robotics and Automation Letters* 10.9 (2025), pp. 9216–9223. DOI: 10.1109/LRA.2025.3592080.

[17] Cheng Hong et al. "The AIzuHand Neuromorphic Prosthetic Hand". In: *Proceedings of the 21st International Conference on Emerging Technologies in Learning, Teaching and Computer Communications (ETLTC)*. January 24–27, 2023. Japan.