



Unified task profiling and performance interference-aware Makespan and energy scheduling for cloud robotics tasks driven by an improved genetic algorithm

Chennian Xiong^a, Weiwei Lin^{a,b} , Huikang Huang^a, Qi Mu^{a,c}, Keqin Li^d

^a Department of Computer Science and Engineering, South China University of Technology, Guangzhou, 510000, China

^b State Key Laboratory of Advanced Papermaking and Paper-based Materials, South China University of Technology, Guangzhou, 510640, China

^c IEIT SYSTEMS Co., Ltd., 801 South Caoshanling Road, Jinan, 250010, Shandong, China

^d Department of Computer Science, State University of New York, NY, 12561, New Paltz, US

ARTICLE INFO

Keywords:

Cloud robotics tasks
Performance interference
Task profiling
Bi-objective optimization
Cloud task scheduling

ABSTRACT

Cloud robotics systems offload heterogeneous, computation- and data-intensive tasks to shared cloud nodes, yet suffer from severe performance interference caused by co-located contention for shared physical resources. Such interference renders idealized execution times unreliable and degrades both Makespan and energy efficiency in conventional schedulers. To enable interference-aware scheduling, it is first necessary to quantitatively characterize the resource demands of diverse tasks and to measure how co-location affects their execution. Therefore, this paper proposes a unified task profiling approach that characterizes CPU, I/O, and memory demands and other key differentiating attributes of eight representative cloud robotics tasks spanning perception, planning, control, and execution. Lightweight, tunable benchmarks are then designed to emulate these resource-access patterns, enabling controlled co-location experiments that construct a quantitative interference-sensitivity matrix. On this foundation, an interference-aware integer programming model is formulated to minimize the weighted sum of Makespan and energy consumption. The main novelty lies in connecting task profiling, benchmark-calibrated interference measurement, and joint Makespan-energy scheduling into a unified interference-aware optimization method. To solve the resulting NP-hard problem, we propose an Improved Genetic Algorithm for Cloud Robotics Task Scheduling (IGA-CTS), which jointly searches task allocation and execution order over the interference-aware scheduling model. In benchmark-calibrated simulation experiments against seven evaluated baselines, no baseline simultaneously achieves lower median Makespan and energy consumption than IGA-CTS at any scale; although some baselines perform better on an individual objective, these advantages are accompanied by inferior performance on the other. An objective-weight sensitivity analysis further shows that this overall trade-off is largely maintained under varying optimization preferences.

1. Introduction

With the evolution of cloud computing toward deep integration with the physical economy, industry-specific clouds have emerged. Cloud robotics, which offloads storage and computation to remote servers, is a representative case due to its high task heterogeneity and stringent reliability requirements. This approach liberates robots from reliance on onboard hardware. By virtualizing robotic functions and computational services, cloud robotics enables on-demand access to cloud resources for perception, decision-making, and execution, improving task efficiency through distributed resources [1]. Compute- or data-intensive

tasks (e.g., path planning, motion simulation, collaborative control, and visual recognition) are offloaded to the cloud, thereby introducing complex scheduling challenges. Unlike general batch processing, multi-dimensional resource contention, specific constraints, and high energy sensitivity need to be considered in cloud robotics scheduling. Thus, scheduling affects not only resource efficiency but also system stability and task quality. Determining the order and placement of tasks on cloud nodes while satisfying relevant constraints is critical for overall efficiency [2].

* Corresponding author.

E-mail addresses: 1119432501@qq.com (C. Xiong), linww@scut.edu.cn (W. Lin), huikanghuang0321@gmail.com (H. Huang), csmuqi@mail.scut.edu.cn (Q. Mu), lik@newpaltz.edu (K. Li).

<https://doi.org/10.1016/j.future.2026.108855>

Received 11 April 2026; Received in revised form 25 September 2026; Accepted 26 September 2026

Available online 1 October 2026

0167-739X/© 2026 Elsevier B.V. All rights reserved, including those for text and data mining, AI training, and similar technologies.

Despite substantial progress in cloud task scheduling, several limitations remain, as emphasized by Alirezazadeh et al. [3]. First, many studies treat cloud robotics tasks as generic computing workloads, resulting in coarse-grained task profiling that fails to capture critical differences in resource demands. A unified theoretical framework integrating diverse task attributes is lacking, which hinders accurate reflection of real-world scheduling scenarios. Second, existing research often focuses on isolated subproblems or specific process links, such as sensor data processing [4], path planning [5], or end-execution scheduling [6], without systematically considering the full cloud robot workflow from perception and decision-making to execution. Third, consideration of performance interference and resource contention remains insufficient. Performance interference refers to the degradation of task execution efficiency (e.g., increased latency or reduced throughput) caused by contention for shared physical resources, such as CPU caches, memory bandwidth, and I/O channels, when multiple tasks are co-located on the same node. Even when overall resource capacity appears adequate, microarchitectural conflicts can severely distort task runtime, making ideal execution durations unreliable for scheduling. Owing to the coarse-grained and fragmented nature of current task analysis, there is no detailed, quantifiable, unified task description that captures heterogeneous resource requirements (CPU, I/O, memory) and dependency relationships across perception, planning, control, and execution. Without such a unified framework, it would also be impossible to quantitatively characterize interference-prone tasks or construct reproducible benchmarks to measure task interference sensitivity, which further prevents schedulers from predicting performance degradation under task co-location. Consequently, research remains confined to specific workloads in isolated resource dimensions, e.g., focusing solely on CPU [7] while neglecting the coupling effects of I/O and memory contention [8]. Finally, many studies are limited to single-objective optimization [9], whereas multi-objective optimization involving metrics such as Makespan and energy consumption faces challenges like solution space explosion and conflicting optimization paths. Traditional approaches often fail to achieve balanced and robust trade-offs in such settings.

To address the aforementioned challenges, this paper develops a systematic approach that integrates unified task profiling, quantitative interference measurement, and interference-aware scheduling optimization. Focusing on the industry-specific scenario of cloud robotics, we present a comprehensive solution that spans from foundational modeling to algorithmic optimization. The scientific contribution of this work is not the isolated use of common evolutionary components such as adaptive genetic operators, elitist retention, or hybrid selection. Instead, this work contributes a unified interference-aware optimization method that links cloud robotics task taxonomy, normalized multi-resource profiles, benchmark-calibrated co-location interference, and allocation-order optimization. The main contributions of this work are threefold:

- (1) We conduct a fine-grained analysis of eight core tasks within a cloud robotics system, creating a unified quantitative profile that captures their heterogeneous characteristics, including resource demands (CPU, I/O, memory), ideal execution durations, and scheduling frequencies. Unlike task abstractions that treat robotic workloads as generic computation jobs, this profile provides a common basic attribute representation across perception, planning, control, and execution tasks. Such fine-grained and unified task profiling is essential for establishing a common foundation that enables accurate interference analysis and robust scheduling decision-making.
- (2) We design and implement a set of lightweight, tunable benchmarks that emulate the resource access patterns of these tasks. By conducting controlled co-location experiments, we construct an interference sensitivity matrix that quantifies the performance degradation for each task pair, providing a measurable

foundation for interference-aware scheduling. This creates a calibration path from task-level resource profiles to pairwise slowdown parameters used by the scheduler, rather than assuming ideal execution times.

- (3) We formulate the scheduling problem as an interference-aware integer programming model to minimize the weighted sum of Makespan and energy consumption, and propose IGA-CTS to solve this NP-hard problem. IGA-CTS jointly searches task allocation and execution order through a two-layer chromosome and evaluates each candidate schedule using interference-aware task durations and utilization-based energy consumption, with adaptive parameter adjustment, elitist retention, and hybrid selection serving as supporting search mechanisms. Simulation experiments on small-, medium-, and large-scale datasets conducted in the benchmark-calibrated simulation environment indicate that IGA-CTS provides a competitive overall Makespan–energy trade-off relative to the evaluated baselines under the default balanced weighting.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 provides a detailed analysis of the system and task profiles for cloud robotics. Section 4 presents the methodology for quantifying performance interference. Section 5 formulates the bi-objective optimization problem and details the proposed IGA-CTS algorithm. Section 6 describes the experimental setup and discusses the results. Finally, we conclude the paper with a summary of contributions and directions for future work.

2. Related work

This section reviews existing literature on cloud robot task research, performance interference, and multi-objective scheduling optimization, and identifies the key limitations that motivate our work.

2.1. Cloud robot task research

Several studies have briefly analyzed the resource requirements of certain cloud robotics tasks. Calderon-Cordova et al. investigated the CPU, memory, and I/O requirements of path planning and actuator control tasks and identified evidence of performance interference through quantitative experiments, though without drawing theoretical conclusions about the interference relationships [5]. Chen et al. indirectly explored the CPU and I/O requirements of partial tasks [10], and Etezadi et al. preliminarily evaluated the computing and communication requirements of various cloud robotics components [11]. However, as noted in the review by Alirezazadeh et al. existing research mostly targets specific scenarios and lacks a unified theory for integrating diverse attributes, while the consideration of performance interference and resource contention remains inadequate [3].

For task-level scheduling, where robots offload tasks to cloud or edge nodes and the focus is on “who performs the computation”, similar to our research, Alirezazadeh et al. proposed GTBA, a dynamic scheduling method based on a full task grid for robot networks with cloud–edge-device collaboration, which minimizes Makespan and ensures load balancing under time window constraints [7]. Wang et al. proposed PMW, a Min-Min-based multi-task scheduling algorithm for cloud-centric batch tasks, which computes task weights from priority and execution time differences and greedily reduces Makespan while improving load balancing and QoS [12]. Both studies, however, abstract cloud robot tasks into general computational workflows, considering only task computation load and node computing capacity, while neglecting co-location performance interference, other resource types, and task dependencies.

At the service level, where robots invoke pre-deployed cloud services, Yin et al. proposed the two-stage service selection strategy DVHGA, which filters candidate services by QoS requirements and

globally optimizes service sequences via a hybrid genetic algorithm to minimize total cost [13]; their follow-up work introduces the evolution factor, hunting factor, and parental similarity mechanisms to improve scheduling efficiency across both small- and large-scale scenarios [9]. Although these studies consider multi-dimensional resource demands, they do not quantify the impact of resource contention on task execution performance with fine granularity.

In industrial cloud robotics scheduling, most research addresses only localized tasks. Xie et al. modeled industrial robot monitoring tasks as workflows and proposed INSGA2-TGHR, an improved NSGA-II based on transcription gene and heuristic recombination, to optimize latency and cost in a cloud-edge collaborative environment [1]; other studies optimize individual subtasks such as real-time sensor data processing [4], path planning [5], or end-effector scheduling [6]. In terms of resource usage modeling, existing studies either consider only computing resources or ignore the impact of I/O resources [8]. Research is still lacking that jointly considers the relative differences in resource attributes among perception, decision-making, planning, and execution tasks, together with their co-located performance interference and integrated scheduling. Recent studies have begun to address more explicit edge-cloud coordination for dependent and data-oriented tasks. Sun et al. formulated latency-aware scheduling for logically related service tasks in a collaborative IoT-edge-cloud network, jointly considering data aggregation, transmission, task processing, and load balancing [14]. Kouloumpiris et al. developed a binary-integer-linear-programming framework for task allocation in an edge/hub/cloud architecture, with latency and energy as alternative optimization objectives and with communication and device constraints explicitly modeled [15]. These studies strengthen the system-level treatment of communication, placement, and energy in distributed computing environments. However, they do not characterize server-internal co-location slowdowns across heterogeneous cloud robotics task categories or connect measured interference sensitivities with allocation and execution-order decisions.

2.2. Performance interference in cloud

Performance interference arises from contention for shared physical resources (CPU, memory, I/O, cache) among co-located virtual machines (VMs) or tasks, leading to performance degradation [16]. Accurate modeling and quantification of interference are therefore critical for designing effective interference-aware scheduling strategies.

Some studies rely on hardware performance counters (HPCs) to collect low-level metrics such as CPU utilization, last-level cache (LLC) miss rate, memory bandwidth, and I/O throughput [16,17], with Lu et al. [18] focusing on interference from I/O-intensive operations and Tzenetopoulos et al. [19] on co-located compute-intensive workloads. Other studies employ benchmarks that simulate resource contention to quantify interference; however, these benchmarks either allow precise control over contention on only a single resource dimension [20], or merely approximate the intended contention intensity by co-running heterogeneous workloads, lacking fine-grained control [21].

Building on such modeling and quantification efforts, interference-aware scheduling research has explored two main strategies. Some studies proactively select deployment conditions that minimize interference; for instance, Shaw et al. combined an application's interference prediction model with energy efficiency optimization to assign tasks to more favorable environments [22]. Others keep interference below a threshold to guarantee basic service quality, as in Nabavinejad et al.'s dynamic interference threshold for QoS-compliant scheduling under Hadoop data skew [23]. Recent work has further connected interference prediction with scheduling and cluster-management decisions in co-located cloud environments. Liu et al. proposed ISACPP, an interference-aware scheduling approach for deep-learning training workloads that predicts co-location performance and uses multi-stage interference quantification to select GPU devices with lower overall

interference [24]. Song et al. presented C-Koordinator, an interference-aware management platform for large-scale co-located microservice clusters that uses CPI-based interference prediction to guide mitigation in production cloud environments [25]. These studies confirm the importance of incorporating interference information into placement or scheduling decisions, but they mainly target generic deep-learning or microservice workloads rather than cloud robotics task profiles, and they do not integrate benchmark-calibrated task-pair slowdowns with allocation-order optimization.

For industrial cloud robot tasks, modeling interference faces unique challenges due to task heterogeneity and varying resource demands. As noted by Alirezazadeh et al. [3], most existing studies focus on specific scenarios rather than establishing a unified theoretical framework, making it difficult to study performance interference and resource contention across all task types in industrial cloud robot systems.

2.3. Multi-objective scheduling

Several meta-heuristic methods have demonstrated strong global search capabilities in high-dimensional, nonlinear, and multi-constrained scheduling problems. Elgendy et al. proposed the parallel distributed genetic algorithm PDGA, combining a mixed-integer programming model with a dynamic task allocation strategy for multi-task scheduling in dynamic environments [26]. Peng et al. designed a two-layer planning model that quantifies device warm-up energy consumption and integrates task cohesion metrics with NSGA-II for energy-efficient service selection and scheduling [27]. Liu et al. developed an improved INSGA-II with a three-level encoding scheme and heuristic initialization, balancing the interests of multiple stakeholders in the integrated scheduling of manufacturing and logistics services [28]. Zhang et al. proposed the three-stage genetic algorithm ISGA, which compresses the search space via K-means clustering and adopts a real-time sequential scheduling strategy for multi-user scenarios [29].

In addition, related cloud-manufacturing studies have explored service-composition optimization, reinforcement-learning, and dynamic decision-making approaches [30–32]. However, for cloud robotics task scheduling, meta-heuristic algorithms are more applicable: they rapidly provide stable, low-overhead, near-optimal solutions for NP-hard problems without requiring large amounts of training data or extensive environment interaction, and can directly model multi-objective trade-offs. In contrast, reinforcement learning suffers from high training costs, slow convergence, and complex reward design, making it difficult to meet the fast and highly reliable scheduling requirements of cloud robotics systems. Recent edge-cloud workflow studies further emphasize the interaction between precedence constraints, communication, and energy. Li investigated energy-constrained DAG scheduling on edge and cloud servers with sequential wireless transmission and overlapped communication and computation [33]. Uma and Shukla proposed an energy- and performance-aware workflow scheduler that combines virtual network function placement and service-function-chain optimization for deadline-sensitive edge-cloud workflows [34]. These approaches primarily model communication, placement, or workflow-level energy constraints, whereas the present work focuses on benchmark-calibrated server-internal interference and its effect on task duration during joint allocation-order optimization.

In summary, existing scheduling studies provide mature optimization operators, whereas cloud robotics task characterization remains largely focused on individual components rather than a unified workflow-level task profile. Existing interference studies have also not connected measurable co-location effects with cloud robotics task categories and allocation-order scheduling decisions. Accordingly, this study proposes a unified interference-aware scheduling method in which task profiles characterize workflow-level resource-demand patterns, benchmarks quantify pairwise co-location slowdowns, and the scheduler jointly optimizes Makespan and energy consumption.

Table 1
Definition of system modeling symbols.

Symbol	Definition
x_{ij}	If task i is assigned to node j , then $x_{ij} = 1$, otherwise $x_{ij} = 0$
t_i	The actual running time of task i
$r_i^{\text{cpu}}, r_i^{\text{io}}, r_i^{\text{mem}}$	The CPU, IO, memory resource requirements of task i
$U_j^{\text{cpu}}(t), U_j^{\text{io}}(t), U_j^{\text{mem}}(t)$	The CPU, IO, and memory utilization of node j at time t
$C_j^{\text{cpu}}, C_j^{\text{io}}, C_j^{\text{mem}}$	The total CPU, IO, and memory resource capacities of node j
N	Node collection
M	Task collection
σ_j	The task execution order on node j
$M_j(t)$	The set of tasks running concurrently on node j at time t
S_i, C_i	The start time and the completion time of task i
P_i	The set of predecessor tasks of task i
B_i	The set of tasks co-located with task i at its start time
p_j^{idle}	The idle power of node j

3. System nodes and task profile

To achieve refined modeling of cloud robot task scheduling in cloud environments, this section characterizes the two core entities of the system: nodes and tasks. Node profiling describes both static resource capacities and dynamic energy consumption behaviors, while task profiling captures resource demands, execution characteristics, dependencies, and scheduling frequency. Unlike existing studies that model a single task or an isolated module, we incorporate the entire cloud robot workflow, spanning perception, decision-making, planning, control, and execution, into a unified framework with a consistent resource demand characterization. Such unified profiling is a prerequisite for interference-aware scheduling: without it, the scheduler cannot identify resource contention relationships among co-located tasks, nor can consistent benchmarks be constructed for measuring interference sensitivity. Accordingly, this section first builds the node-side resource and energy profile (Section 3.1) and then the task-side profile (Section 3.2). The symbols used in system modeling are listed in Table 1.

3.1. Node profile analysis

In a cloud computing environment, cloud robot tasks are scheduled and executed through the coordinated operation of multiple computing nodes, each with limited CPU, I/O, and memory resources; both task runtime quality and system energy consumption depend on how these resources are utilized.

Each node j in the node set $N = \{N_1, N_2, \dots, N_n\}$ represents a resource pool with three core capabilities: CPU, I/O, and memory, denoted as $C_j = \{C_j^{\text{cpu}}, C_j^{\text{io}}, C_j^{\text{mem}}\}$. Task execution must strictly satisfy the resource constraints in Eqs. (14)–(16); for instance, a path-planning task with very high CPU demand (Table 2) cannot be executed on a node with insufficient CPU capacity. Beyond static capacity, node profiling must also capture the impact of utilization variation: under high utilization, contention degrades performance and sharply increases energy consumption, whereas excessively low utilization wastes resources and energy.

Accordingly, each node is modeled as an integrated entity governed jointly by resource capacity, utilization rate, and energy consumption, which provides both the capacity constraints for the scheduling model and the foundation for the joint Makespan–energy optimization objective [9]. Energy consumption in cloud systems is primarily driven by resource utilization [35] and consists of a static idle component and a dynamic component that increases linearly with the utilization rate. In

this section, the dynamic energy consumption is modeled as follows: Let $\ell \in \mathcal{R} = \{\text{cpu}, \text{io}, \text{mem}\}$ denote a resource dimension.

$$E_j^\ell(t) = \alpha_\ell \cdot U_j^\ell(t), \quad (1)$$

where $E_j^\ell(t)$ denotes the dynamic energy consumption of resource ℓ on node j at time t , α_ℓ is the power coefficient of resource ℓ , and $U_j^\ell(t)$ represents the utilization rate of resource ℓ at time t , defined as:

$$U_j^\ell(t) = \frac{\sum_{i \in M_j(t)} r_i^\ell}{C_j^\ell}, \quad (2)$$

where C_j^ℓ is the capacity of resource ℓ on node j , and $M_j(t) = \{i \mid x_{ij} = 1, S_i \leq t < C_i\}$ is the set of tasks running concurrently on node j at time t , where S_i and C_i denote the start and completion times of task i . Tasks assigned to the same node may execute concurrently: a task occupies its full demand vector r_i throughout its active interval $[S_i, C_i]$, so at any instant the utilization of each resource dimension is the aggregate normalized demand of the tasks running at that instant. The total dynamic energy consumption of node j at time t , denoted as $E_j(t)$, is expressed as:

$$E_j(t) = E_j^{\text{cpu}}(t) + E_j^{\text{io}}(t) + E_j^{\text{mem}}(t), \quad (3)$$

where $E_j^{\text{cpu}}(t)$, $E_j^{\text{io}}(t)$ and $E_j^{\text{mem}}(t)$ are the energy consumptions of node j at time t respectively. The total energy consumption E_{total} of all nodes in the system over the entire scheduling period is:

$$E_{\text{total}} = E_{\text{idle}} + \sum_{j \in N} \int_0^{T_{\text{Makespan}}} E_j(t) dt, \quad (4)$$

where E_{idle} is the basic energy consumption, and T_{Makespan} is the total time of the entire scheduling period. Specifically, $E_{\text{idle}} = \sum_{j \in N} p_j^{\text{idle}} \cdot T_{\text{Makespan}}$, i.e. every node is charged its idle power over the whole scheduling horizon, and α_ℓ in Eq. (1) is expressed in watts, denoting the dynamic power drawn by resource ℓ at full utilization. Since $M_j(t)$ changes only at the task start and completion times of a given schedule, $U_j^\ell(t)$ is piecewise constant and the integral in Eq. (4) is evaluated over the corresponding time segments. The Makespan is defined as the latest completion time among all tasks:

$$T_{\text{Makespan}} = \max_{i \in M} C_i, \quad C_i = S_i + t_i. \quad (5)$$

3.2. Task profile analysis

3.2.1. Task characteristic description

Unlike traditional approaches that describe tasks only by name or coarse-grained categories, this study constructs a unified full-process profiling framework covering perception, decision-making, planning, control, and execution. Features of all task modules are extracted and quantitatively mapped into a homogeneous attribute set, specifically CPU, I/O, and memory demands, ideal execution duration t_{i0} , and scheduling frequency f_i , to ensure dimensional consistency for subsequent interference analysis and scheduling modeling. Let $M = \{M_1, M_2, \dots, M_m\}$ denote the task set, where each task is characterized by a resource demand vector $r_i = \{r_i^{\text{cpu}}, r_i^{\text{io}}, r_i^{\text{mem}}\}$; all attributes are quantified on a unified numerical scale of 1–10, determined by task complexity, computational intensity, data volume, operational steps, and empirical time proportions. This unified profiling serves two key purposes. First, it integrates task descriptions originally scattered across different functional modules into a single modeling framework, avoiding the fragmentation caused by describing perception, control, and planning tasks separately. Second, and more importantly, it provides a unified resource requirement benchmark for the interference sensitivity measurement in Section 4.4: only when the CPU, I/O, and memory requirements of tasks are uniformly quantified can comparability among tasks be ensured in interference test benchmarks, and the impact of co-location on execution time be measured.

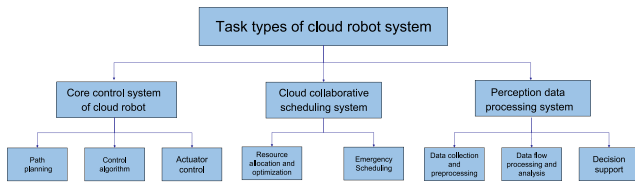


Fig. 1. Cloud robotics tasks.

Since existing research mostly addresses individual components in isolation, we categorize the system modules into three main classes, as shown in Fig. 1: robot core control systems, cloud collaborative scheduling systems, and perception data processing systems, which are further subdivided into eight functional task types [10,11,36,37], including data acquisition and processing [38], decision support [39], path planning [40], and robot task execution [38]. Their profiling attributes are analyzed one by one below.

As the core component of the robotic system, the cloud robotic core control system is responsible for motion control and real-time decision-making to ensure that the robot adheres to the planned paths and motions. The task modules of this system are mainly divided into the following three categories:

- **Path planning (PP):** Based on the robot's current pose, target coordinates, and environmental obstacle information, it solves for the optimal motion trajectory using cloud computing power. CPU demand is very high due to frequent spatial searches and constraint-condition calculations; memory demand is low, primarily for caching the current local map and transient intermediate path-planning results; I/O demand is very low, as each invocation exchanges only compact planning inputs (the current pose, target, and relevant local obstacle descriptors) and the resulting path, while map data are not retransmitted in full. Since it is triggered only upon task issuance, robot repositioning, or environmental changes, its frequency is low; however, owing to its high computational complexity and potentially intricate algorithmic logic, each execution instance is comparatively time-consuming.
- **Actuator control (AC):** As the core execution link in the cloud-end control closed loop, it receives the instructions generated by the control algorithm (e.g., motor torque, joint angles, and action-timing parameters), performs instruction parsing, verification, and execution-logic optimization in the cloud, and issues execution signals to the robot terminal. CPU demand is medium for instruction parsing and constraint verification; I/O demand is medium to guarantee communication between the cloud and terminal devices; memory demand is very low, since it caches only lightweight actuator status parameters, safety thresholds, and a small instruction execution log, without retaining historical states or intermediate results. Its frequency is synchronized with the cloud control algorithm, and the duration of a single execution is comparable to the control cycle.
- **Control algorithm (CA):** Employing algorithms such as PID and model predictive control, it generates control instructions by processing sensor feedback data and conducting error-compensation calculations in the cloud. This requires extensive computation and logical judgment over input parameters and feedback, resulting in high CPU demand and a relatively long execution duration; I/O demand is medium for hardware communication, and memory demand is very high for storing historical parameters and accumulated errors. Its frequency is relatively high, typically slightly lower than that of data-collection tasks, since it must operate promptly on newly collected data to maintain process stability and precision.

The cloud-end collaborative scheduling system is responsible for the global coordination and optimized allocation of cloud computing

resources (CPU, memory, and I/O) as well as emergency handling. It constitutes the core of cloud computing support for the efficient operation of cloud robotic systems. This system consists of the following two main task modules:

- **Resource allocation and optimization (OR):** Based on the system's scheduling requirements and the current load of cloud resources, it allocates computing resources and task priorities through optimization algorithms (e.g., genetic algorithms or greedy strategies). Its computational complexity is moderate, yet it must handle numerous factors and access resource data extensively, resulting in relatively long processing times; I/O and memory demands are medium, mainly for storing multi-resource states and allocation rules. Unlike high-frequency data or control tasks, it is triggered only by scheduling plan adjustments, major equipment changes, or preset scheduling interval modifications, so its frequency is low.
- **Emergency scheduling (ES):** When abnormalities occur in the robot system (e.g., sensor failures, path blockages, or resource overloads), it rapidly adjusts cloud resource allocation and task execution sequences to maintain the continuity of core functions. The urgent regeneration of scheduling plans imposes substantial CPU demand for timely computation; frequent reading of current state data and writing of new plans generates significant I/O operations; memory demand is low, primarily associated with short-lived system-state records and temporary scheduling data during plan regeneration. Although its occurrence is rare, once triggered it involves a large volume of complex work; therefore, its frequency is set to the lowest level and its duration to the longest.

The perception data processing system is responsible for collecting, processing, and analyzing various types of data in the robot's working environment, providing data support for the above two parts of the system. The system mainly includes the following three task modules:

- **Data collection and preprocessing (DCP):** It performs high-frequency collection of environmental and equipment status data via onboard visual, force, and lidar sensors, followed by preliminary cloud-end processing including filtering, denoising, and format standardization. Computational complexity is moderate, involving mainly data cleaning and format conversion; I/O demand is medium per instance, supporting the transmission and temporary storage of the current batch of sensor data; memory demand is low, requiring only caching of a small amount of raw data and preprocessing results. Each cycle is short (duration value 2), while its execution frequency is the highest of all tasks.
- **Data stream processing and analysis (DPA):** It performs in-depth processing and analysis on the collected real-time data streams to extract valuable information (such as environmental trends and equipment status). It requires real-time handling of multiple data streams and complex analyses (e.g., data mining and real-time statistics), resulting in a relatively long duration and demanding very high CPU computing capacity. Due to the heavy read/write load from processing large volumes of multi-sensor data, I/O demand is also very high. Large amounts of data streams and intermediate analysis results must be cached, leading to very high memory usage. Although real-time processing is required, its processing interval is slightly longer than that of data collection and direct control tasks; hence, frequency is set to medium.
- **Decision support (DS):** Based on the results of data flow analysis, it provides decision recommendations using complex data analysis models and algorithms, which are relatively time-consuming and thus demand high CPU processing capability. Its I/O demand is moderate, with data primarily sourced from sensors and existing analysis results; memory demand is low, as it primarily retains decision models and final recommendations drawn from existing analysis results, rather than storing raw or intermediate data. It is triggered only when decisions are needed (e.g., plan formulation, strategy adjustment, or handling major issues), so its frequency is relatively low.

Table 2
Resource requirements for each task category.

Type of task	CPU demand	I/O demand	Memory demand	Ideal duration	Occurrence frequency
PP	9	2	4	4	2
AC	5	5	1	3	8
CA	8	5	10	3	8
OR	5	5	6	5	4
ES	7	8	3	8	1
DCP	5	5	3	2	10
DPA	9	9	10	5	6
DS	8	5	3	6	4

Based on the above analysis, the relevant attributes of the eight task types are expressed using normalized ordinal scores from 1 to 10, with higher values indicating greater demand. These scores are not derived from a single production trace and do not represent exact hardware measurements such as CPU cycles, I/O bandwidth, or memory capacity; instead, they describe typical category-level resource-demand patterns of representative cloud robotics tasks. This relative profiling design is adopted because, as further discussed in Section 6.1, publicly available ROS traces and robotic benchmarks do not provide a single consistently labeled source covering all eight task categories together with the resource, dependency, co-location-interference, and energy information required by this study.

Specifically, we first extract the dominant resource characteristics of each task type from prior studies [10,11,36–40] and from its functional role in the cloud robotics workflow. CPU demand is jointly reflected by algorithm time complexity and execution computation volume; I/O demand is determined by data read/write frequency and device communication volume; and memory demand is determined by data cache volume, intermediate-result storage, and parameter persistence. These qualitative characteristics are mapped to an ordinal scale: 1–2 for very low demand, 3–4 for low demand, 5–6 for medium demand, 7–8 for high demand, and 9–10 for very high or bottleneck-level demand. Adjacent values within each range are assigned through pairwise cross-comparisons among task categories.

For example, PP and DPA receive the highest CPU scores because PP requires intensive spatial search and constraint solving, whereas DPA performs large-scale stream analysis. DPA receives the highest I/O score because it continuously processes multi-sensor streams. CA and DPA receive the highest memory scores because CA retains historical parameters and accumulated errors, whereas DPA caches substantial intermediate results. AC receives the lowest memory score because it retains only lightweight actuator-status parameters and safety thresholds. For ideal duration and occurrence frequency, when explicit values are unavailable in the literature, the former is determined with reference to task complexity, computation volume, and resource demand, whereas the latter is related to trigger frequency, task dependency, and environmental changes. The three resource levels, the ideal duration, and the occurrence frequency jointly characterize a task type and should not be read in isolation: DCP, for example, has a medium persistence I/O level, yet its highest occurrence frequency and shortest ideal duration lead to substantial repeated I/O activity.

The demand levels are physically realized in the controlled benchmark configurations of Section 4.4. However, Table 2 remains an internally consistent and configurable workload-profile template for interference measurement and scheduling optimization, rather than trace-specific empirical constants. The scores can be recalibrated using scenario-specific knowledge, monitoring data, or real ROS deployment traces without changing the overall modeling framework.

3.2.2. Task dependency

Cloud robotics tasks typically exhibit clear dependency relationships. These dependencies primarily fall into two categories: data dependencies and control dependencies. A data dependency arises when

the output of a predecessor task serves as the required input for a successor task. For example, a planning task can only generate an execution path once it receives environmental information output by a perception task. Representative cases include:

- DCP to PP: The PP generates motion trajectories using the environmental perception data (e.g., obstacle coordinates and robot pose) provided by the DCP; any delay in the DCP directly affects the start time of the PP.
- DPA to DS: The DS relies on the real-time feature analysis results (e.g., equipment status trends and anomaly detection metrics) output by the DPA to produce visualization reports.
- DCP to DPA: The DPA must first clean and structure the raw data collected by the DCP before performing subsequent streaming computations.

Control dependencies, by contrast, require that a successor task wait for the predecessor task to complete before it can start. For instance, certain control tasks can only drive the actuators once parameter computation is finished. Typical cases include:

- CA to AC: The AC performs real-time actuation using the dynamic adjustment parameters (e.g., motor torque and joint angles) output by the CA; the completion time of the CA determines the earliest possible start time of the AC.
- DPA to ES: The ES is triggered only when the DPA detects an abnormal event (e.g., equipment failure or resource bottleneck), at which point the DPA must provide a completion signal (event-driven).

In the actual production environment, when some tasks arrive, due to cases where the preceding tasks have already been completed, there is no need to consider the complete preceding dependencies anymore, and there will only be partial dependencies, which are constrained by Eq. (17).

4. Quantifying performance interference

In practice, it is difficult to obtain real-world cloud robotics task instances; therefore, to simulate their resource contention, we design a set of controllable benchmarks. Accordingly, in this section, we first provide the definition of performance interference, then design an interference quantification benchmark capable of accurately regulating the intensity of multi-dimensional resource contention, and finally present the quantitative results of interference sensitivity for cloud robotic tasks.

4.1. Definition of performance interference

Cloud robotics systems offload tasks to shared cloud nodes that use virtualization to provide on-demand resources. However, co-located virtual machines still contend for host physical resources, such as CPU time slices, LLC, memory bandwidth, and I/O channels, and thus cannot run as ideally as in an exclusive environment. The resulting fluctuations in workload execution efficiency and service quality, defined as performance interference [16], render scheduling algorithms designed under idealized assumptions ineffective in practice, as their preset resource allocation strategies and optimization objectives cannot be reliably met. We quantify the interference-induced performance loss as the relative difference between a task's completion time under co-located conditions and that under ideal isolated execution, as formulated in Eq. (6):

$$I(A \& bg) = \frac{T_{A \& bg} - T_A}{T_A}, \quad (6)$$

where A is the benchmark under test, bg is the background benchmark, $I(A \& bg)$ is the performance interference sensitivity when benchmark A and benchmark bg are run co-located, $T_{A \& bg}$ is the task execution time when benchmark A and benchmark bg are run co-located, and T_A is the task execution time when benchmark A is run alone. We use this concept to evaluate the actual running time of tasks in the co-location

running scenario. When task i runs with the co-located load bg , its actual running time is:

$$t_i = S_{i \& bg} \cdot t_{i0}, \quad (7)$$

where $S_{i \& bg}$ is the performance sensitivity when task i and load bg are run co-located, and t_{i0} is the running time of task i under the ideal condition. When task i has multiple co-runners on its assigned node, we use the dominant-interferer rule $S_{i \& bg} = 1 + \max_{b \in B_i} I(i \& b)$, where $B_i = M_j(S_j) \setminus \{i\}$ and $S_{i \& bg} = 1$ if $B_i = \emptyset$. Tasks that start at the same instant on the same node are included in each other's co-runner sets, while tasks linked by dependency constraints never execute concurrently and therefore never inflate each other's durations. The resulting interference-aware durations are used in Eqs. (5) and (4). This start-time dominant-interferer approximation keeps slowdown evaluation within the pairwise calibrated sensitivity values and avoids iterative recomputation of co-runner sets during population-level search.

4.2. Analysis of interference factors

In cloud data centers, CPU, I/O, and memory resources are generally unsaturated; hence, although sufficient capacity prevents blocking caused by resource scarcity, co-located tasks still interfere with each other through conflicting cross-accesses to microarchitecture-level shared resources (e.g., LLC, memory bandwidth), independently of overall resource saturation. This section analyzes CPU and I/O interference factors for all tasks, while memory interference analysis is restricted to CA, OR, and DPA, as only these tasks exhibit significant memory demands.

All eight task categories exhibit varying degrees of CPU demand: even when cores are sufficient, frequent context switching degrades performance, and contention for the shared L3 cache increases miss rates and, consequently, memory accesses. All task categories except PP have at least medium I/O demand: concurrent I/O operations on the same file system trigger lock contention and delayed requests, while I/O-intensive tasks incur frequent system calls and context switches, with tight bandwidth further exacerbating delays. Beyond same-resource contention, high-I/O tasks frequently enter kernel mode whereas high-CPU tasks consume more user-mode execution time; these distinct demand patterns can imbalance the scheduler's policy, thereby amplifying interference.

Regarding memory access interference, although memory capacity is assumed sufficient, insufficient bandwidth still increases latency; even with adequate bandwidth, NUMA effects (access latency varying with the CPU core's location relative to memory modules) and DDR4/DDR5 memory bank conflicts (the memory controller queues simultaneous requests to the same bank) introduce additional access delays. Moreover, such interference is not confined to memory intensive tasks: high CPU and high I/O tasks also access memory and can interfere mutually, for example, high I/O tasks occupying the memory bus force CPU computations to wait, while CPU tasks occupying scheduling resources slow the storage I/O processing of I/O tasks.

4.3. Scope of the interference model

The interference factors and sensitivity matrix focus on server-internal contention, covering CPU, cache, memory bandwidth, NUMA, storage I/O, and device I/O. These effects can be measured in a controlled manner via the co-running benchmarks. Network bandwidth contention and transmission latency follow different mechanisms, depending on time-varying link quality, routing, and competing flows, and are not faithfully captured by node-level benchmarks. This separation aligns with Ref. [41], which models communication and computing separately; network effects are thus handled at the communication layer, not in the node-level sensitivity matrix.

This work focuses on cloud-side scheduling, where IGA-CTS optimizes server-side execution under co-location interference for off-loaded tasks. However, end-to-end completion also includes uplink data transmission and downlink result return. Bandwidth contention and transmission latency may therefore increase Makespan beyond the optimized server-side execution. Given the distinct modeling assumptions required for communication dynamics, a joint communication-computation extension of the proposed model is left as future work.

4.4. Design of benchmarks for measuring interference sensitivity

Different from iBench and FECBench mentioned in related work [20, 21], we design a performance interference sensitivity testing benchmark that can accurately control the intensity of multi-dimensional resource contention. According to the resource requirements in Table 2, we configure resource-access simulations for different task types, which can run either in isolation or co-located with other tasks; the interference sensitivity of the task under test is then obtained via Eq. (6).

CPU-contention bench. Simulate the computational behavior of tasks at different CPU demand levels (1–10), reproducing CPU time-slice contention and cache conflicts through controllable arithmetic operations.

- Implementation mechanism: Fixed-point matrix multiplication is adopted to avoid the hardware acceleration associated with floating-point operations. The matrix dimension $N \times N$ is mapped to the CPU demand level via

$$N = k \cdot \lceil S_{\text{cpu}} \cdot \delta \rceil, \quad (8)$$

where S_{cpu} is the CPU demand level (1–10) of the task in Table 2, $k = 128$ is the base dimension coefficient, and $\delta = 0.8$ is the calibration factor, determined through preliminary experiments to ensure a linear correlation $R^2 \geq 0.92$ between CPU utilization and demand level.

- Thread isolation strategy: The POSIX thread library (*pthread*) isolates benchmark threads from system threads and binds them to specific CPU cores, eliminating interference from OS scheduling.
- Cache contention simulation: Matrix data-block sizes are controlled to span the L1/L2/L3 cache levels. For example, high-CPU-demand tasks (e.g., PP and DPA) set the block size to 1.2 times the L3 capacity, triggering cache misses and reconstruction.

I/O-contention bench. Simulate the I/O behavior of tasks at different I/O demand levels (1–10), covering both storage I/O and device-communication I/O contention.

- A mixed sequential/random read/write model is employed, with I/O intensity mapped to the demand level via request size (B), request frequency (F), and read/write ratio (R):

$$I/O_{\text{intensity}} = \lambda \cdot B \cdot F \cdot [0.7R + 0.3(1 - R)], \quad (9)$$

where $\lambda = 10^{-6}$ is the normalization coefficient, $B \in [4 \text{ kB}, 64 \text{ MB}]$ (corresponding to demand levels 1–10), $F \in [10 \text{ Hz}, 1000 \text{ Hz}]$, and R is configured per task type (e.g., $R = 0.8$ for ES to simulate random state reads during emergency scheduling).

- Delay and bandwidth control: The Linux *ioctl* interface sets the block-device scheduler to *noop* to avoid interference from scheduling algorithms, and *nanosleep* simulates device waiting delays, with duration positively correlated with the demand level (1 ms for level 1, 50 ms for level 10).
- Lock contention reproduction: Concurrent multi-threaded reads/writes to the same file descriptor trigger file-system lock contention, simulating I/O blocking under co-located deployment.

Mem-contention bench. Simulate the memory access behavior of tasks at different memory demand levels (1–10), reproducing bandwidth contention, NUMA access differences, and memory bank conflicts.

- A large-scale contiguous/non-contiguous array access model is adopted, with array capacity (C) and stride (S) mapped to the demand level via

$$C = \mu \cdot S_{\text{MEM}}, \quad (10)$$

$$S = \left\lfloor \frac{\text{page_size} \cdot \theta}{S_{\text{MEM}}} \right\rfloor, \quad (11)$$

where $\mu = 512\text{MB}$ is the base capacity coefficient, S_{MEM} is the memory demand level from Table 2, page size = 4 kB, and $\theta = 8$. High-memory-demand tasks (e.g., CA and DPA) use non-contiguous access (stride $S = 8$) to trigger TLB misses and bank conflicts.

- Memory bandwidth control: The perf tool monitors bandwidth utilization to ensure linear matching between benchmark consumption and memory demand level (level 10 corresponds to $\geq 80\%$ occupancy).
- NUMA architecture adaptation: The benchmark supports binding memory accesses to a specified NUMA node, simulating cross-node memory access latency in cloud servers and ensuring the realism of interference measurements.

4.5. Measurement of task interference sensitivity

Next, the lightweight benchmarks are used to simulate co-located task execution under 1000, 2000, and 3000 access operations. For each test task A , the ideal execution time is first obtained by running each configuration in isolation. Pairwise co-location experiments are then conducted with tasks B , C , and D (using the same number of access operations) to obtain the time differences. For the remaining seven tasks, the identical procedure is repeated. Experiments are performed on a host with Intel Xeon Silver 4110 CPU @ 2.10 GHz (32 logical CPUs), and L3 cache of 11 264 kB. Each configuration is repeated 30 times, and the median value is recorded to eliminate outliers.

Regardless of the number of co-scheduling runs (1000, 2000, or 3000), in the majority of co-scheduling cases, the task performance sensitivity metric $I(A\&bg)$ (as defined in Eq. (6)) remains stable within a range of ± 0.15 around a certain value, with only occasional outliers. This indicates that, under conditions of sufficient resources, the performance interference stays within a bounded range. We define this bounded range as the co-scheduling performance sensitivity of the task, as illustrated in Fig. 2. The heatmap is consistent with the previous analysis: co-scheduling tasks that both exhibit high demand for the same resource (e.g., CPU or I/O) tend to cause significant interference; pairing a high-CPU-demand task with a high-I/O-demand task also induces noticeable interference; moreover, co-scheduling tasks that have high demands for both types of resources further exacerbate this cumulative effect, resulting in a substantial increase in interference sensitivity.

It should be noted that the quantitative interference matrix may be drastically altered on newer hardware platforms. For instance, ARM-based edge servers, with their distinct cache hierarchies and big.LITTLE core asymmetries, can significantly modify interference patterns for CPU- and memory-intensive tasks. More importantly, for perception data processing tasks such as DCP and DPA that are frequently off-loaded to GPU-accelerated nodes, the interference landscape differs markedly due to CUDA/stream-level contention, tensor-core resource sharing, and host-device memory transfer overheads [42]. These differences highlight the importance of context-specific recalibration of the sensitivity matrix on target architectures, without altering the core interference characterization framework proposed in this work.



Fig. 2. Thermal map of performance interference sensitivity.

5. IGA-CTS

5.1. Formalization of the optimization objectives

The cloud robotics task scheduling problem proposed in this paper is a multi-objective integer programming problem aiming to minimize both energy consumption and task completion time (Makespan). Specifically, given a task collection, node resource state $C = \{C_1, C_2, \dots, C_n\}$, and task resource demands and dependency relationships Dependency , we seek an optimal task allocation scheme and execution order $\sigma = \{\sigma_1, \sigma_2, \dots, \sigma_n\}$ to optimize the weighted sum of energy consumption and normalized Makespan, formulated as:

Given : $N, M, \text{Dependency}$,

Find : $M \rightarrow N, \sigma$,

Minimizing : $Z = w_1 \cdot T_{\text{Makespan}} + w_2 \cdot E_{\text{total}}$. (12)

Due to differing magnitudes between the two objectives, normalization is required to prevent dominance by either term. This objective function is also subject to some constraints: for example, each task must be assigned to a node:

$$\sum_{j \in N} x_{ij} = 1, \forall i. \quad (13)$$

There are also the node resource constraints mentioned in Section 3.1. In the proposed formulation, Eqs. (14)–(16) are used as assignment-level resource-admission constraints on the task set assigned to each node. This conservative condition implies instantaneous feasibility since $M_j(t) \subseteq \{i \mid x_{ij} = 1\}$, and it guarantees $U_j^\ell(t) \leq 1$ in Eq. (2). These constraints are expressed as:

$$\sum_i x_{ij} \cdot r_i^{\text{cpu}} \leq C_j^{\text{cpu}}, \forall j, \quad (14)$$

$$\sum_i x_{ij} \cdot r_i^{\text{io}} \leq C_j^{\text{io}}, \forall j, \quad (15)$$

$$\sum_i x_{ij} \cdot r_i^{\text{mem}} \leq C_j^{\text{mem}}, \forall j, \quad (16)$$

In addition, there is the task dependency constraint mentioned in Section 3.2.2:

$$C_i \leq S_k, \quad \forall (i, k) \in \text{Dependency}, \quad (17)$$

where task i is the preceding task of task k , and the start time S_k of task k must exceed the end time C_i of task i .

Under this execution semantics, tasks assigned to the same node may execute concurrently rather than strictly sequentially. A task starts when all its predecessors have completed, i.e., $S_i = \max\{C_p \mid p \in P_i\}$ with zero used for tasks without predecessors, and $C_i = S_i + t_i$. Therefore, dependent tasks never overlap even if they are assigned to the same node, while independent tasks with overlapping active intervals on the same node may cause co-location interference. The execution order σ_j provides a dependency-feasible sequence for each node and a tie-breaking order among tasks whose predecessors are satisfied at the same time.

Remark 1. The formulation in Eqs. (12)–(17) targets the aggregate Makespan–energy trade-off and does not by itself provide hard-deadline guarantees, yet it accommodates deadline enforcement in two complementary ways. (i) A deadline-feasibility layer can be added: for each critical task i with an application-specified deadline D_i , a schedule is accepted only if its predicted completion time C_i satisfies $C_i \leq D_i$ (or, in a soft variant, violations are penalized in the fitness function). (ii) At deployment, tasks with overly stringent deadlines can be executed locally or on nearby edge nodes, while the rest follow the proposed scheduler, consistent with deterministic offloading frameworks that prioritize bounded-deadline completion over latency minimization [43]. A more detailed treatment of deadline-constrained scheduling is omitted due to space limitations and left for future work.

Remark 2. The proposed models involve deliberate simplifications that keep them measurable and tractable, and their limitations should be noted. For the interference model, the sensitivity matrix is calibrated pairwise, and Eq. (7) applies a start-time dominant-interferer approximation; when three or more tasks are co-located, cumulative slowdown may exceed the maximum pairwise effect, and co-runner changes during execution are not re-evaluated. Moreover, the matrix is measured on a single server platform and is hardware-dependent, so it should be recalibrated on target architectures (Section 4.5), and it covers node-internal contention only, excluding network effects (Section 4.3). For the energy model, Eqs. (1)–(4) assume that dynamic power grows linearly with utilization and that idle power is constant over the scheduling horizon; real servers may exhibit nonlinear power behavior due to frequency scaling and thermal effects, and communication energy between robots and the cloud is not modeled. These simplifications preserve reproducibility and fair comparison under identical conditions, and all parameters can be recalibrated from platform-specific measurements (Section 6.1); more fine-grained power and interference modeling is left to future work.

5.2. Overview of scheduling algorithm

To address the problem formulated in Section 5.1, we design the system framework illustrated in Fig. 3. Users submit tasks to the scheduling platform, which determines task-node assignments and execution orders through two modules: the resource allocation module enforces the resource constraints and optimizes the energy consumption defined in Section 3.1, while the task execution order module enforces the dependency constraints of Section 3.2.2, organizes dependency-feasible co-located execution, and performs the performance interference evaluation of Section 4. The problem is solved by the genetic algorithm [44]; given its complexity and bi-objective nature, we propose IGA-CTS, an improved genetic algorithm based on multi-layer encoding and adaptive parameter adjustment, and enhanced with elitism and hybrid selection strategies to effectively balance global search capability and local optimization efficiency.

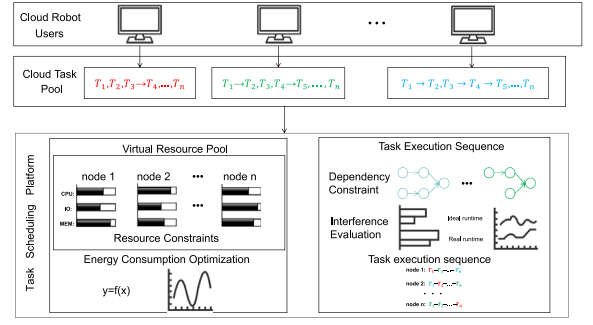


Fig. 3. System framework of the scheduling approach.

5.3. Task scheduling optimization driven by improved genetic algorithm

5.3.1. Multi-layer encoding design and fitness function

Multi-layer encoding design. The single-layer encoding in the basic algorithm optimizes task allocation and execution order simultaneously, which may fail to satisfy multiple constraints and produce invalid solutions; with n nodes and m tasks, its search space is $O(n^m \cdot m!)$. We therefore adopt a two-layer encoding that decomposes the search space into an allocation layer $O(n^m)$ and an order layer $O(m!)$, reducing complexity through layering. The first layer, task allocation encoding, is an array whose length equals the number of tasks, where each element $a_i \in \{1, 2, \dots, n\}$ denotes the node to which task M_i is allocated; this layer satisfies the resource constraints and optimizes energy consumption. The second layer, task execution order encoding, represents the execution sequence of tasks on each node, e.g., $\sigma_j = \{o_1, o_2, \dots, o_{m_j}\}$, where m_j denotes the number of tasks assigned to node j ; task o_1 executes first on node j , followed by o_2 , and so on. This layer satisfies the dependency constraints, ensuring that preceding tasks complete before their dependent successors start, and records a dependency-feasible sequence with a tie-breaking order among tasks whose predecessors are satisfied at the same time.

Fitness function. The objective function in Eq. (12) is used directly as the minimization fitness of an individual:

$$F(Ind) = Z(Ind), \quad (18)$$

where Ind denotes an individual encoding a candidate schedule, and $Z(Ind)$ denotes the objective value of the schedule encoded by Ind . The weights in Eq. (12) satisfy $w_1 + w_2 = 1$; thus, a smaller fitness value indicates a better individual. In fitness evaluation, start and completion times follow the execution model of Section 5.1: $S_i = \max\{C_p \mid p \in P_i\}$ and $C_i = S_i + t_i$. The set B_i of tasks co-located with task i at its start time is obtained from the active tasks on the same node, and the interference-aware duration $t_i = S_{i \& B_i} \cdot t_{i0}$ follows Eq. (7). The Makespan is $T_{Makespan} = \max_{i \in M} C_i$ (Eq. (5)), and total energy is accumulated from the resource utilization over time according to Eq. (4). Tasks starting at the same instant on the same node are included in one another's co-located task sets.

5.3.2. Genetic operations

Selection operation. IGA-CTS adopts a hybrid selection strategy that combines multiple selection methods to enhance population diversity, avoid premature convergence, and balance global exploration and local exploitation. The selected strategies are as follows:

- Roulette wheel selection: For the minimization fitness in Eq. (18), an inverse score $q_k = 1/(F_k + \epsilon)$ is first computed for individual k , and the selection probability is $p_k = q_k / \sum_i q_i$. A random number $r \in [0, 1]$ is generated, and the individual satisfying $\sum_{m=0}^{k-1} p_m < r \leq \sum_{m=0}^k p_m$ is selected, ensuring lower-fitness individuals dominate reproduction.

- **Tournament selection:** Randomly selects a fixed number of individuals from the population each time, choosing the lowest-fitness one to enter the next generation. This process repeats until the new population reaches the target size, promoting retention of high-quality individuals.
- **Rank-based selection:** Individuals are ranked by ascending fitness, and selection probabilities are assigned based on ranks rather than absolute values, mitigating bias from large fitness disparities and preserving diversity.

Each strategy has distinct advantages: roulette wheel excels in early exploration, tournament selection accelerates late-stage convergence, and rank-based selection stabilizes performance under extreme fitness distributions. Basic genetic algorithms employ a single strategy, while hybrid strategies leverage complementary strengths to adapt to complex problems and reduce reliance on a single strategy. For example, switching between roulette wheel and tournament selection every S steps can maintain diversity while accelerating convergence. However, hybrid strategies also have drawbacks. Different selection strategies may favor different individuals, and strategy conflicts may lead to confusion in the evolutionary direction. Frequent switching may disrupt the structure of well-established genes and hinder convergence. Therefore, a stage-based strategy may be more effective, such as using roulette wheel selection for the first 30% of iterations and tournament selection for the remaining 70%, balancing exploration and exploitation while avoiding disturbances caused by frequent switching.

Crossover and mutation operation. Crossover is performed separately on the two encoding layers. The allocation layer uses multi-point crossover, which exchanges multiple gene segments between parents and generates more diverse offspring than single-point crossover, helping escape local optima under complex resource constraints. The order layer uses PMX: since execution sequences must satisfy dependency constraints, standard crossover may produce invalid permutations, whereas PMX preserves legality by exchanging parental segments and resolving conflicts through a mapping table, while retaining high-quality gene sequences to accelerate convergence. Mutation is likewise layer-specific: the random mutation in the basic algorithm ignores the characteristics and optimization direction of the current solution, potentially causing ineffective mutations; we therefore swap task positions on the order layer and adjust task-node assignments on the allocation layer under the premise of maintaining legal solutions, enhancing population diversity and reducing futile mutations.

To further balance exploration and exploitation, the crossover rate is set high in early generations to promote diversity and gradually decreased to preserve good solutions, and the mutation rate is likewise reduced over generations to refine the search; both rates follow a simple linear decay from their initial values.

5.3.3. Elitist retention strategy

This subsection describes the elitist retention strategy adopted in IGA-CTS. In the experimental section, we compare multiple retention rates, including static retention rates and a dynamic retention rate that decreases linearly over generations. The dynamic retention rate is defined in Eq. (21), where the initial retention rate $init$ gradually decreases to the final rate fin as the generation number increases, thereby balancing early-stage preservation of high-quality solutions with late-stage maintenance of population diversity.

Elite individuals may be disrupted by crossover or mutation operations, particularly in dynamic multi-objective optimization where random operations often degrade high-quality solutions (i.e., those with low fitness values). To prevent the genetic algorithm from losing the global optimal solution, IGA-CTS uses elitist retention. In each iteration, the optimal individuals in the current population are retained and directly transferred to the next generation. Let the optimal individual sets in the current population be A^* and S^* , with fitness value F^* . In each iteration, a certain proportion of optimal individuals are copied

to the next generation without participating in selection, crossover, or mutation:

$$A^{(g+1)} = A^{(g)} \cup A^*, \quad (19)$$

$$S^{(g+1)} = S^{(g)} \cup S^*. \quad (20)$$

By retaining the best solutions, this strategy prevents the loss of optimal solutions during iterations and ensures a gradual improvement in solution quality.

5.4. Algorithm design

The pseudocode of the proposed IGA-CTS algorithm is presented in Algorithm 1. The population is first initialized with random allocation and order encodings, and each individual is evaluated by the fitness function (Lines 1–4). In each generation, parents are selected via the hybrid strategy–roulette wheel selection for the first 30% of generations and tournament selection for the remaining 70%, followed by crossover and mutation applied separately on the allocation and order layers with adaptively adjusted rates (Lines 5–7). Offspring violating the constraints in Eqs. (13)–(17) are assigned infinite fitness, while feasible individuals are evaluated for interference-aware Makespan and energy consumption (Lines 8–16). Elitist retention is then applied and the best solution is updated (Lines 18–19). After G_{max} generations, the algorithm returns the optimal solution, including task-node assignments and execution sequences (Lines 20–21). The time complexity is $O(G_{max} \times NP \times D)$, where NP is the population size and D the task count, with space complexity $O(NP \times D)$.

Algorithm 1: Improved Genetic Algorithm for Cloud Robotics Task Scheduling (IGA-CTS)

Input: M (tasks), N (nodes), G_{max} , NP **Output:** Best solution (task allocation and execution order)

```

1: Initialize population  $Pop$  of size  $NP$  with random allocation and
   order encoding
2: for each individual  $Ind \in Pop$  do
3:   Evaluate  $Fit(Ind)$  ▷ Eq. (12)
4: end for
5: for  $g \leftarrow 1$  to  $G_{max}$  do
6:   Select parents using hybrid strategy
7:   Apply crossover and mutation separately on allocation and
   order layers
8:   for each individual  $Ind$  in offspring do
9:     if Violation of Eqs. (13)–(17) then
10:       $Fit(Ind) \leftarrow \infty$ 
11:     else
12:       Resolve  $S_i$  and  $B_i$  by discrete-event evaluation of the
       candidate schedule
13:        $t_i = S_{i\&bg} \cdot t_{i0}$  ▷ Eq. (7)
14:       Update  $C_i$  and  $T_{Makespan}$  ▷ Eq. (5)
15:       Update  $E_{total}$  ▷ Eq. (4)
16:     end if
17:   end for
18:   Elitist retention ▷ Eqs. (19)–(20)
19:   Update  $BestSolution \leftarrow \arg \min(Fit(Pop))$ 
20: end for
21: return  $BestSolution$ 

```

6. Experiment

6.1. Experimental configuration

In this work, we validate the proposed algorithm through simulation experiments. The simulation code, implemented in Python, is deployed on a host with an Intel(R) Xeon(R) Gold 5218 CPU @

2.30 GHz and 64 GB RAM, and defines task attributes (CPU, I/O, and memory demands, ideal execution duration, performance sensitivity, and dependency relationships) as well as node resource capacities. The task generation module creates a heterogeneous task pool from predefined templates, determines task density and quantity according to the occurrence frequency attributes, and ensures the validity of dependency relationships. The node generation module produces allocatable nodes with appropriate resource capacities to guarantee compliance with resource constraints, while the scheduling module simulates performance interference caused by co-located execution and reflects it in actual execution durations. The scheduling module follows the execution model of Section 5.1 and evaluates interference-aware durations and energy using Eqs. (7) and (4). Node capacities are generated in the same per-dimension resource units as the task demand scores in Table 2. Accordingly, the demand terms in Eqs. (2) and (14)–(16) are dimensionally consistent with the corresponding capacities. Section 4.4 provides the benchmark configurations for the demand levels, which were physically executed on the measurement server described in Section 4.5 to obtain the interference sensitivity values. For deployment on a target platform, task resource demands and node capacities should be calibrated from platform-specific measurements. The calibration should measure per-task CPU, I/O, and memory demands, determine the available node capacities, and remeasure co-location interference and energy behavior under the target workload. These platform-specific parameters can then be used to instantiate the scheduling model for that platform. To assess the algorithm's robustness and scalability across problem scales, experiments are carried out on datasets of three scales, following the setup of existing work [45–47]: a small-scale dataset (100 tasks, 10 nodes), a medium-scale dataset (1000 tasks, 50 nodes), and a large-scale dataset (5000 tasks, 100 nodes).

This experimental setup validates IGA-CTS in an empirically parameterized simulation environment rather than a full physical cloud robotics deployment. This scope is mainly due to the difficulty of building a reproducible robot-cloud testbed in which heterogeneous tasks, co-location interference, resource contention, and energy consumption can all be controlled and repeatedly measured. Existing ROS-based traces and robotic benchmarks typically support component-level performance analysis, sensor replay, or navigation evaluation, but do not directly provide the complete scheduling information required here, including task dependencies, multi-resource demands, co-location interference measurements, and energy observations. The experiments therefore focus on controlled validation of the proposed interference-aware scheduling mechanism, while real ROS 2/Gazebo traces and physical cloud robotics deployment are left for future work. Accordingly, the comparative and deployment-related conclusions in this paper are scoped to the benchmark-calibrated simulation environment and should not be interpreted as physical deployment validation.

6.2. Experimental schemes

The experimental scheme examines interference perception, elite retention under different parameters, configuration of selection strategies, comparison with existing methods, and objective-weight sensitivity. Each algorithmic variant is independently run 30 times. The median of the results under the objective in Eq. (12) is used for comparison, and the half interquartile range (half-IQR) is reported as the fluctuation range for the repeated-run comparative experiments. For the normalized objective values reported in Section 6.3.5, the two terms of Eq. (12) are normalized by fixed scale-specific constants taken from the median Makespan and energy consumption of IGA-CTS under the default balanced setting (w_1, w_2) = (0.5, 0.5); the same constants are used for all algorithms and weight settings on that scale. The Makespan and total energy consumption for each experimental component are analyzed in Section 6.3.

The parameter settings in the following experimental schemes follow common configurations in relevant research [44,48]. For example,

the 30%/70% two-phase hybrid selection strategy is designed based on the typical convergence characteristics of this scheduling problem: the large early-stage solution space requires strong exploration, while the later stage demands intensive exploitation to balance the conflicting Makespan and energy consumption objectives. This ratio is a commonly used, experimentally validated proportion for balancing exploration and exploitation in most problem instances; supplementary experiments are additionally conducted for scenarios where this ratio is suboptimal to explore suitable proportions. For the elite retention strategy, elite retention prevents the loss of high-quality solutions, yet an excessively high ratio reduces population diversity and causes premature convergence, while an overly low ratio fails to effectively inherit superior genes. The fixed ratios of 5% and 10% are widely adopted benchmarks in classical literature, and the dynamic decreasing schemes (a higher early-stage ratio to protect initially discovered high-quality solutions, gradually reduced to enhance later-stage exploration) are specially designed to address the “path conflict” in bi-objective optimization. These schemes better adapt to the distinct optimization paths of Makespan (tending to temporal convergence) and energy consumption (favoring resource balancing); the two dynamic curves (10%→5% and 20%→5%) respectively cover mild and aggressive decrement modes, enabling systematic comparison.

For examining interference awareness effects, we compare two IGA-CTS variants across the three scales, both using roulette wheel selection for the first 30% of generations and tournament selection for the remaining 70%: the interference-aware (Inf) variant accounts for the impact of performance interference on task duration, while the non-interference (No-Inf) variant does not. The two variants are compared in terms of Makespan and total energy consumption.

For validating elitist retention effects, experiments are conducted under IGA-CTS with identical settings except for the retention strategy: no retention, 5% retention, 10% retention, and dynamic retention decreasing from 10% to 5% and from 20% to 5%, all compared in terms of Makespan and total energy consumption. The dynamic retention rate is defined as:

$$elite = \max \left(fin, init - (init - fin) \times \frac{g}{G_{max}} \right), \quad (21)$$

where *elite* represents the current generation's elite retention rate, *init* the initial rate, *fin* the final rate, *g* the current generation number, and G_{max} the total number of generations.

For validating different hybrid selection configurations, we designed five methods. GA_base uses only tournament selection. GA_WP_R_3 and GA_SR_3 combine tournament, roulette wheel, and best selection, selecting one strategy every 5 generations with probabilities 0.6/0.2/0.2 and with equal probability, respectively. GA_SR_2 and GA_IS_2 combine tournament and roulette wheel selection: the former randomly selects between the two with equal probability every 5 generations, while the latter switches by iteration stage, applying roulette wheel selection for the first 30% of generations and tournament selection for the remaining 70%. Subsequent experiments further examine GA_IS_2 under different stage division settings.

To validate the IGA-CTS algorithm, we configure its parameters according to the best experimental results obtained from the three scales (small, medium, large) in the preceding experiments, and then compare it with seven existing algorithms from three families: genetic algorithm based schedulers (CMfg-SCCCS, DISML, PDGA), reinforcement learning based schedulers (PPO, A2C), and swarm-intelligence hybrid schedulers (PSO-ACO, WOA-PSO):

CMfg-SCCCS [49]: This paper uses [0,1] encoding instead of multi-layer encoding and only employs roulette wheel selection strategy instead of hybrid selection, with single-point crossover and single-point mutation in both crossover and genetic operations.

DISML [28]: This paper designs specific crossover and mutation operations including order-preserving crossover, two-point crossover, insertion mutation, and random mutation to enhance search capability and solution diversity.

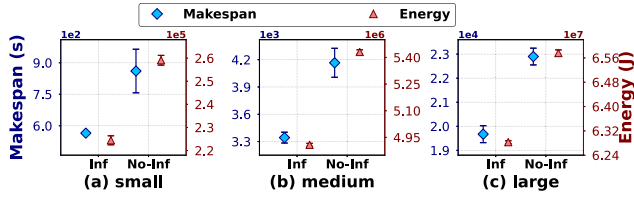


Fig. 4. Comparison of optimization effects between Inf and non-Inf.

PDGA [26]: This paper divides the task population into subpopulations evolving independently across computing threads, with individual evaluation executed in a multithreaded pool; it introduces a “Drains between Grains” strategy that scans idle resource periods after allocation to optimize execution order, together with multi-point crossover (OBX) and swap mutation.

PPO [50]: This reinforcement-learning baseline learns task-node assignment policies through a scalarized Makespan-energy reward and updates the scheduling policy using clipped policy optimization.

A2C [51]: This actor-critic reinforcement-learning baseline jointly learns an allocation policy and a value estimator under the same scalarized scheduling reward.

PSO-ACO [52]: This hybrid swarm-intelligence baseline combines particle-swarm search for global allocation exploration with ant-colony-style pheromone guidance for refining scheduling decisions.

WOA-PSO [53]: This hybrid swarm-intelligence baseline combines whale-optimization exploration with particle-swarm updating to search the combinatorial allocation-order space.

To further support the comparisons in Fig. 8, we performed pairwise statistical analyses between IGA-CTS and each competing algorithm for every scale and metric based on the corresponding 30-run results. Because the distributions of stochastic metaheuristic results are not assumed to be normal, the two-sided Mann-Whitney U test was adopted for independent pairwise comparisons. The resulting p -values were adjusted using the Holm correction within each scale-metric comparison family. We also report Cliff’s delta as a non-parametric effect size and 95% normal-approximation bootstrap confidence intervals (CIs) for the median differences, where the difference is defined as IGA-CTS minus the competing method. Unless otherwise stated, the default objective-weight setting for the comparative experiments is $(w_1, w_2) = (0.5, 0.5)$.

To examine whether the weighted-sum formulation produces stable and tunable scheduling decisions under different optimization preferences, we further conduct an objective-weight sensitivity analysis for IGA-CTS and two key baselines, CMfg-SCCCS and DISML. Five weight settings are tested: $(w_1, w_2) = (0.1, 0.9), (0.3, 0.7), (0.5, 0.5), (0.7, 0.3)$, and $(0.9, 0.1)$, where w_1 and w_2 denote the weights of Makespan and energy consumption, respectively. The resulting Makespan, energy consumption, and normalized scalar objective Z are reported for the small-, medium-, and large-scale datasets. CMfg-SCCCS and DISML are selected because they are the two baselines that are strongest on an individual objective in the main comparison: CMfg-SCCCS is competitive in energy consumption, whereas DISML is competitive in Makespan.

6.3. Experimental results

6.3.1. Evaluating the optimization effect of interference awareness

As shown in Fig. 4, the interference-aware method outperforms the non-aware method in terms of Makespan and energy consumption across all three scales. This is because the interference-aware approach accounts for the impact of co-location on task execution durations, which not only introduces unpredictability to Makespan but also complicates energy estimation. Co-location schemes optimized

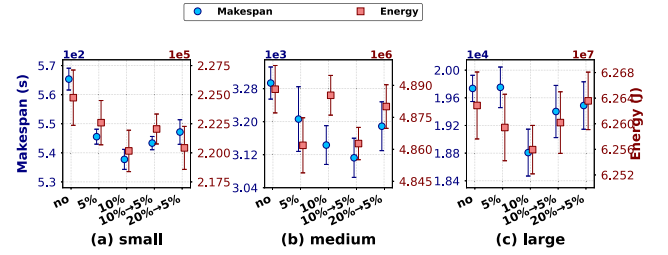


Fig. 5. The effect of elite retention strategy under different parameter settings.

under ideal task durations may become suboptimal in real-world scenarios due to interference-induced delays, and the same applies to energy consumption. Additionally, the non-aware method fails to account for excess energy consumption caused by prolonged execution times. Experimental results confirm that the interference-aware method optimizes task-node assignments and execution sequences to mitigate these effects.

6.3.2. Evaluation of elitism strategies under different parameter settings

As shown in Fig. 5, for small-scale problems, retaining 10% of the best individuals yields the lowest observed median objective value among the tested retention settings. The limited solution space allows rapid convergence to low-Makespan solutions with balanced resource allocation and minimal energy consumption. A 5% retention rate retains insufficient elites, maintaining higher population diversity but slower convergence, though still superior to no retention. Dynamic strategies initially accelerate convergence with higher retention but later enhance exploration by reducing retention. However, in small-scale settings, the benefits of dynamic adjustment are overshadowed by population instability caused by retention rate fluctuations, making static 10% retention preferable.

For medium-scale scenarios, dynamic retention decreasing from 10% to 5% yields the lowest observed median objective value among the tested retention settings. Early-stage high retention narrows the search space, while later reductions boost diversity, balancing local and global search to minimize Makespan and energy consumption. The inferior performance of dynamic 20%-to-5% retention is consistent with premature convergence to local optima due to excessive early retention, limiting later-stage exploration.

In large-scale scenarios, retaining 10% of individuals yields the lowest observed median objective value among the tested retention settings, while dynamic retention strategies appear less stable, indicating that linear decay is not universally beneficial in complex search spaces. As task scale grows, high-quality allocation-order structures demand longer preservation; reducing retention in later generations risks weakening their inheritance before convergence. Moreover, scalarized fitness (combining Makespan and energy) can amplify objective competition under changing retention rates, biasing search toward one objective. Hence, the final configuration adopts the static 10% retention for stability. Future work may combine interference-aware scheduling with Pareto-based selection (e.g., non-dominated sorting with diversity preservation) to enhance search diversity in large-scale bi-objective optimization. This component-level observation does not undermine IGA-CTS’s overall effectiveness: with the validated retention and other enhancements, the complete algorithm still provides a competitive overall Makespan-energy trade-off among the compared methods in the large-scale scenario (Section 6.3.4).

6.3.3. Effects of different scheme settings in selection strategy

The results of the five methods across the three scales are illustrated in Fig. 6. In the small-scale cases, GA-IS_2 gives the best observed objective values: within the limited solution space, the staged use of roulette wheel selection (emphasizing diversity) for the first 30% of

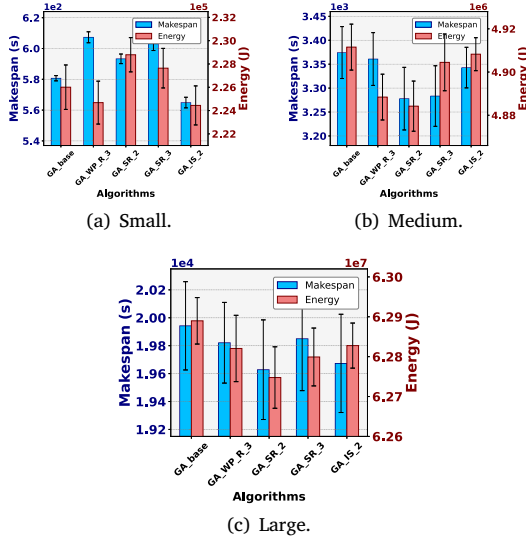


Fig. 6. Effects of different scheme settings in selection strategy.

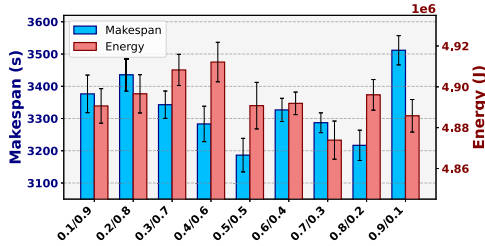


Fig. 7. The performance of GA_IS_2 algorithm under different stage parameter settings.

generations and tournament selection (emphasizing convergence) for the remaining 70% enables a rapid balance between exploration and exploitation, whereas the frequent strategy switching of the other hybrids does not leverage multi-strategy advantages and instead induces population oscillations and an unstable evolutionary process. In the medium- and large-scale cases, GA_SR_2 gives the best observed objective values: as the search space grows, switching strategies every few generations continuously balances exploration and exploitation, avoids premature convergence, and adapts to the problem's characteristics at different generations, improving adaptability and robustness. Using more strategies, however, does not necessarily yield better results: GA_WP_R_3 and GA_SR_3 combine roulette wheel (diversity), tournament (convergence), and best selection (elitism), but alternating them can produce conflicting selection pressures that prevent the formation of a cohesive optimization process.

As shown in Fig. 6(b), GA_IS_2 does not exhibit its usual advantage in the medium-scale scenario, performing only marginally better than GA_base: the medium-scale solution space is neither small enough for a fixed division to dominate nor large enough to benefit clearly from staged exploration, and the default 30%/70% division may be unsuitable. We therefore compared different phase divisions, with results shown in Fig. 7. Under the 50%/50% division, Makespan is lowest observed and energy consumption is second best, indicating a favorable observed trade-off between the two objectives; under the 70%/30% division, energy consumption is lowest observed while Makespan ranks third, as the higher exploration proportion identifies low-energy solutions with acceptable Makespan. In contrast, the 40%/60% division yields the highest observed energy consumption (insufficient exploration and excessive exploitation fragment resources

and destabilize solution quality), while the 90%/10% division yields the highest observed Makespan: since the optimization pathways of energy consumption and Makespan are inherently separated, excessive early exploration prevents sufficient convergence in later stages, favoring energy optimization at the expense of Makespan.

6.3.4. Performance comparison with existing methods

The performance comparison of IGA-CTS against seven baselines across three scales under the default balanced weighting is presented in Fig. 8, where bars and error bars denote the median and fluctuation range over 30 independent runs. No baseline simultaneously achieves lower median Makespan and energy consumption than IGA-CTS at any scale; every method that excels in one objective performs worse on the other, indicating that IGA-CTS provides a competitive overall Makespan–energy trade-off under the default setting.

The inferential statistics further support this observation. The complete pairwise results are reported in Table 3, including the Holm-adjusted p -value, Cliff's delta, and the 95% normal-approximation bootstrap confidence interval (CI) of the median difference for every scale-metric pair. Among the 42 pairwise comparisons, 41 remain statistically significant after Holm correction ($p < 0.05$); the only non-significant comparison is the large-scale Makespan comparison against DISML ($p = 0.52$). The results also clarify the single-objective exceptions: DISML has a lower Makespan median at all three scales, with significant differences at the small and medium scales, whereas CMfg-SCCCS has lower energy medians at the medium and large scales. These advantages are accompanied by substantially worse performance on the other objective. Thus, under the default balanced weighting, no baseline shows a statistically significant advantage over IGA-CTS on both objectives at any scale, supporting the conclusion that IGA-CTS provides a competitive overall Makespan–energy trade-off.

DISML attains the lowest median Makespan at all three scales under the default setting, although its large-scale difference from IGA-CTS is not statistically significant, and it incurs substantially higher energy consumption. This pattern suggests that its adaptive hybrid operators overly prioritize temporal optimization, leading to task concentration on high-performance nodes and uneven utilization. Conversely, CMfg-SCCCS yields slightly lower energy at the medium and large scales, yet its single-layer [0,1] encoding and roulette-only selection weaken global exploration, producing conservative allocations that considerably prolong Makespan. PDGA performs worst on both objectives at all scales: its parallel subpopulations introduce substantial overhead, and its complex operators generate numerous invalid solutions under resource and dependency constraints, rendering it unsuitable for the studied scenarios.

The reinforcement learning methods PPO and A2C achieve intermediate solution quality, comparable to swarm-intelligence hybrids yet consistently inferior to IGA-CTS, because their policies converge to suboptimal regions under the scalarized bi-objective reward within a limited training budget; they also incur the highest runtime (Table 4). PSO-ACO and WOA-PSO exhibit low runtime, but their swarm updates suffer from premature convergence in the combinatorial allocation-order space, leaving their solution quality clearly behind IGA-CTS across all scales.

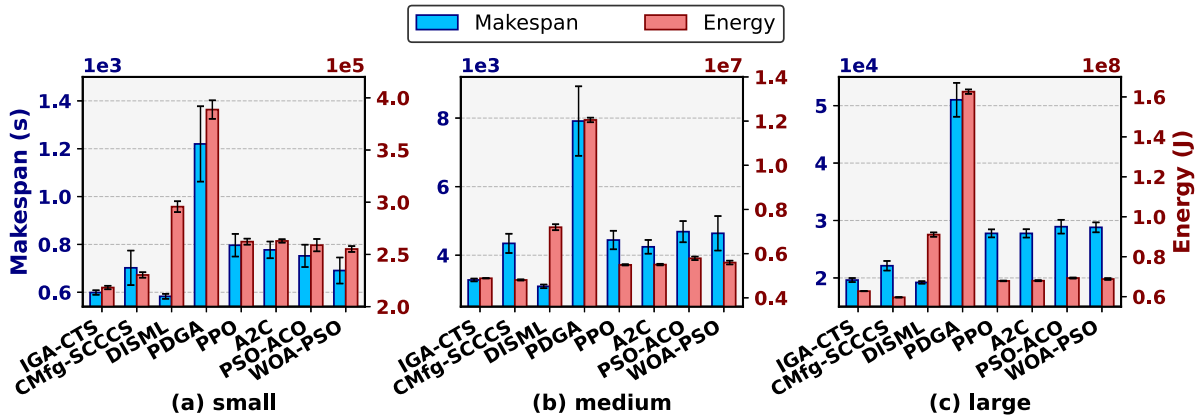
Table 4 reports the actual runtime of each algorithm on the large-scale dataset (5000 tasks, 100 nodes), where each value is the median over 30 independent runs, consistent with the statistical protocol of the other experiments. The runtime of IGA-CTS is consistent with its time complexity $O(G_{\max} \times NP \times D)$: it is considerably lower than that of the reinforcement learning methods, where policy training dominates, and that of PDGA, although higher than that of the lightweight heuristics, which in turn sacrifice solution quality. Since cloud robotics schedulers typically re-optimize pooled tasks periodically at scheduling epochs on the order of minutes, completing a full optimization for 5000 tasks within about four minutes is feasible for dynamic runtime scheduling; moreover, fitness evaluation of the population is independent across individuals and can be parallelized on multi-core servers to further reduce wall-clock time.

Table 3

Complete pairwise statistical comparison between IGA-CTS and the competing algorithms over 30 independent runs.

Scale	Baseline	Metric	Median IGA	Median baseline	Δ	Holm p	Cliff's δ	95% CI
Small	CMfg-SCCCS	Makespan	599.00	702.20	-103.20	2.11×10^{-10}	-1.000	[-140.90, -65.50]
Small	CMfg-SCCCS	Energy	218 289.00	230 338.71	-12 049.71	1.77×10^{-5}	-0.691	[-17 361.00, -6737.00]
Small	DISML	Makespan	599.00	583.00	16.00	0.0117	0.380	[3.60, 28.40]
Small	DISML	Energy	218 289.00	295 747.57	-77 458.57	2.11×10^{-10}	-1.000	[-81 501.37, -73 415.77]
Small	PDGA	Makespan	599.00	1 220.00	-621.00	2.11×10^{-10}	-1.000	[-740.40, -501.60]
Small	PDGA	Energy	218 289.00	388 728.15	-170 439.15	2.11×10^{-10}	-1.000	[-177 843.65, -163 034.65]
Small	PPO	Makespan	599.00	796.70	-197.70	2.96×10^{-10}	-0.980	[-229.40, -166.00]
Small	PPO	Energy	218 289.00	262 243.07	-43 954.07	0.00288	-0.480	[-74 595.28, -13 312.86]
Small	A2C	Makespan	599.00	777.22	-178.22	2.96×10^{-10}	-0.980	[-202.82, -153.62]
Small	A2C	Energy	218 289.00	262 921.86	-44 632.86	1.11×10^{-8}	-0.900	[-69 455.36, -19 810.36]
Small	PSO-ACO	Makespan	599.00	752.20	-153.20	6.68×10^{-9}	-0.900	[-186.25, -120.15]
Small	PSO-ACO	Energy	218 289.00	258 879.12	-40 590.12	1.11×10^{-8}	-0.900	[-51 657.12, -29 523.12]
Small	WOA-PSO	Makespan	599.00	690.95	-91.95	5.42×10^{-8}	-0.837	[-113.45, -70.45]
Small	WOA-PSO	Energy	218 289.00	255 193.84	-36 904.84	0.0170	-0.360	[-72 293.84, -1515.84]
Medium	CMfg-SCCCS	Makespan	3 277.95	4 345.00	-1 067.05	2.40×10^{-8}	-0.858	[-1 393.00, -741.00]
Medium	CMfg-SCCCS	Energy	4 884 214.69	4 808 254.48	75 960.21	0.00667	0.409	[12 566.00, 139 354.00]
Medium	DISML	Makespan	3 277.95	3 091.90	186.05	6.36×10^{-5}	0.602	[75.45, 296.65]
Medium	DISML	Energy	4 884 214.69	7 195 468.47	-2 311 253.78	2.11×10^{-10}	-1.000	[-2 388 233.00, -2 234 274.56]
Medium	PDGA	Makespan	3 277.95	7 893.47	-4 615.52	2.11×10^{-10}	-1.000	[-5595.77, -3635.27]
Medium	PDGA	Energy	4 884 214.69	12 052 986.58	-7 168 771.89	2.11×10^{-10}	-1.000	[-7 360 600.60, -6 976 943.18]
Medium	PPO	Makespan	3 277.95	4 057.26	-779.31	2.11×10^{-9}	-0.940	[-894.50, -664.10]
Medium	PPO	Energy	4 884 214.69	5 965 668.95	-1 081 454.26	2.11×10^{-9}	-0.940	[-1 280 354.26, -882 554.26]
Medium	A2C	Makespan	3 277.95	4 244.68	-966.73	2.11×10^{-10}	-1.000	[-1072.47, -860.99]
Medium	A2C	Energy	4 884 214.69	5 503 462.99	-619 248.30	6.00×10^{-9}	-0.900	[-650 336.40, -588 160.20]
Medium	PSO-ACO	Makespan	3 277.95	4 686.62	-1 408.67	8.00×10^{-9}	-0.900	[-1618.97, -1198.37]
Medium	PSO-ACO	Energy	4 884 214.69	5 789 599.70	-905 385.01	6.00×10^{-9}	-0.900	[-974 043.41, -836 726.61]
Medium	WOA-PSO	Makespan	3 277.95	4 638.83	-1 360.88	8.00×10^{-9}	-0.900	[-1650.78, -1070.98]
Medium	WOA-PSO	Energy	4 884 214.69	5 595 948.80	-711 734.11	2.11×10^{-10}	-1.000	[-781 086.60, -642 381.60]
Large	CMfg-SCCCS	Makespan	19 627.97	22 119.10	-2 491.13	4.88×10^{-8}	-0.850	[-3454.50, -1527.70]
Large	CMfg-SCCCS	Energy	62 747 633.68	59 660 000.00	3 087 633.68	2.11×10^{-10}	1.000	[2 913 140.00, 3 262 127.00]
Large	DISML	Makespan	19 627.97	19 212.75	415.22	0.52	0.098	[-211.00, 1041.44]
Large	DISML	Energy	62 747 633.68	91 071 182.16	-28 323 548.48	2.11×10^{-10}	-1.000	[-29 182 975.00, -27 464 121.96]
Large	PDGA	Makespan	19 627.97	51 015.12	-31 387.15	2.11×10^{-10}	-1.000	[-33 306.95, -29 467.35]
Large	PDGA	Energy	62 747 633.68	162 633 987.45	-99 886 353.77	2.11×10^{-10}	-1.000	[-100 500 246.40, -99 272 486.20]
Large	PPO	Makespan	19 627.97	27 756.53	-8 128.56	2.50×10^{-8}	-0.880	[-10 473.00, -5784.00]
Large	PPO	Energy	62 747 633.68	67 878 370.30	-5 130 736.62	1.00×10^{-8}	-0.880	[-5 322 657.08, -4 938 816.16]
Large	A2C	Makespan	19 627.97	27 755.65	-8 127.68	2.50×10^{-8}	-0.880	[-11 113.00, -5143.00]
Large	A2C	Energy	62 747 633.68	67 982 684.49	-5 235 050.81	1.00×10^{-8}	-0.880	[-5 409 749.41, -5 060 352.21]
Large	PSO-ACO	Makespan	19 627.97	28 920.43	-9 292.46	1.34×10^{-8}	-0.900	[-11 826.00, -6759.00]
Large	PSO-ACO	Energy	62 747 633.68	69 323 804.47	-6 576 170.79	2.11×10^{-10}	-1.000	[-6 845 823.49, -6 306 518.09]
Large	WOA-PSO	Makespan	19 627.97	28 806.93	-9 178.96	6.04×10^{-7}	-0.771	[-12 601.00, -5757.00]
Large	WOA-PSO	Energy	62 747 633.68	68 836 672.72	-6 089 039.04	2.11×10^{-10}	-1.000	[-6 406 852.30, -5 771 225.80]

Δ is the median difference, computed as IGA-CTS minus the baseline; negative values indicate a lower IGA-CTS median. Cliff's δ is computed as $P(\text{IGA-CTS} > \text{baseline}) - P(\text{IGA-CTS} < \text{baseline})$; positive values indicate that the baseline is stochastically smaller (better) on the listed metric. The 95% CIs are normal-approximation bootstrap intervals for Δ , computed as $\Delta \pm 1.96 SE_{boot}$.

**Fig. 8.** Performance comparison with multiple existing algorithms.**Table 4**

Execution runtime of all algorithms on the large-scale dataset.

Method	IGA-CTS	CMfg-SCCCS	DISML	PDGA	PPO	A2C	PSO-ACO	WOA-PSO
Runtime (s)	226.48	213.17	49.86	272.11	479.16	501.12	54.27	29.32

Table 5
Objective-weight sensitivity analysis with two key baselines.

Scale	w_1	w_2	IGA-CTS			CMfg-SCCCS			DISML		
			Makespan	Energy	Z	Makespan	Energy	Z	Makespan	Energy	Z
Small	0.1	0.9	647.38	2.1631×10^5	0.9999	738.42	2.2938×10^5	1.0690	591.57	2.9488×10^5	1.3145
Small	0.3	0.7	603.74	2.1739×10^5	0.9995	721.68	2.2981×10^5	1.0984	587.51	2.9533×10^5	1.2413
Small	0.5	0.5	599.00	2.1829×10^5	1.0000	702.20	2.3034×10^5	1.1137	583.00	2.9575×10^5	1.1641
Small	0.7	0.3	576.73	2.1958×10^5	0.9758	689.91	2.3092×10^5	1.1236	579.96	2.9622×10^5	1.0849
Small	0.9	0.1	557.29	2.2065×10^5	0.9384	671.73	2.3147×10^5	1.1153	576.11	2.9659×10^5	1.0015
Medium	0.1	0.9	3352.47	4.8712×10^6	0.9999	4487.86	4.7985×10^6	1.0211	3128.79	7.1531×10^6	1.4135
Medium	0.3	0.7	3291.82	4.8752×10^6	1.0000	4416.59	4.8034×10^6	1.0926	3111.58	7.1682×10^6	1.3121
Medium	0.5	0.5	3277.95	4.8842×10^6	1.0000	4345.00	4.8083×10^6	1.1550	3091.90	7.1955×10^6	1.2082
Medium	0.7	0.3	3208.94	4.8878×10^6	0.9855	4281.97	4.8133×10^6	1.2101	3078.47	7.2175×10^6	1.1007
Medium	0.9	0.1	3143.12	4.8913×10^6	0.9631	4217.68	4.8190×10^6	1.2567	3062.86	7.2434×10^6	0.9892
Large	0.1	0.9	19715.34	6.2709×10^7	0.9999	22269.07	5.9612×10^7	0.9685	19258.36	9.0826×10^7	1.4008
Large	0.3	0.7	19639.12	6.2724×10^7	0.9999	22134.81	5.9647×10^7	1.0037	19235.15	9.0963×10^7	1.3088
Large	0.5	0.5	19627.97	6.2748×10^7	1.0000	22119.10	5.9660×10^7	1.0389	19212.75	9.1071×10^7	1.2151
Large	0.7	0.3	19341.83	6.2769×10^7	0.9899	21976.95	5.9689×10^7	1.0692	19189.81	9.1284×10^7	1.1208
Large	0.9	0.1	18967.25	6.2791×10^7	0.9698	21852.21	5.9713×10^7	1.0972	19156.02	9.1517×10^7	1.0242

6.3.5. Objective-weight sensitivity analysis

Table 5 reports the sensitivity results for IGA-CTS and the two key baselines under five weight settings. The results first show that IGA-CTS responds to the weight changes in the expected direction on all three dataset scales: as the Makespan weight increases, the obtained Makespan consistently decreases while energy consumption increases, and the opposite tendency is observed when the energy weight dominates. These results indicate that the objective weights act as actual search preferences rather than only as post-hoc reporting coefficients: assigning a larger weight to Makespan guides IGA-CTS toward shorter completion times, whereas assigning a larger weight to energy consumption tends to preserve lower energy usage. Compared with the two key baselines, IGA-CTS obtains the lowest normalized objective value Z for all five weight settings on the small- and medium-scale datasets and for four of the five weight settings on the large-scale dataset. In pairwise terms, IGA-CTS achieves a lower Z in 29 of the 30 comparisons against CMfg-SCCCS and DISML. IGA-CTS obtains a lower Z than DISML in all 15 settings: DISML has a shorter Makespan in 12 of these settings, but its energy consumption is substantially higher in every setting, and in the remaining three settings IGA-CTS is better on both raw metrics. CMfg-SCCCS is competitive in energy-oriented cases and obtains a lower Z value only on the large-scale dataset under $(w_1, w_2) = (0.1, 0.9)$; however, this gain is accompanied by considerably longer Makespan. Overall, the sensitivity analysis supports both the tunability of IGA-CTS and the robustness of its Makespan–energy trade-off across most tested preferences.

6.3.6. Scalability and deployment considerations

The experiments evaluate IGA-CTS as a centralized scheduler for a cloud-side task pool, consistent with the practice of one scheduler per resource pool. Scalability within this scope is mainly determined by the problem size, and the computational cost is highly correlated with the number of tasks. However, in future large-scale cloud robotics and IoT–edge–cloud continuum environments, many geographically distributed devices and edge domains may generate or be assigned tasks. Maintaining a single global task pool and evolving complete scheduling solutions globally may impose high computational and coordination overhead, requiring architectural solutions.

Inspired by scalable deterministic offloading architectures [41] where distributed subnetworks manage local tasks to avoid overloading a single scheduler, IGA-CTS can be extended hierarchically. Specifically, domain-level schedulers run the proposed interference-aware scheduling independently on their own task pools, with the same task profiling and interference models, while an upper-level coordinator handles only cross-domain offloading, load balancing, and cloud admission. This design offers two benefits: each domain-level scheduler solves a smaller subproblem with a reduced task count, and domain

schedulers can run in parallel. Thus, the global problem decomposes into smaller coordinated subproblems without altering the scheduling model. Developing and validating this distributed implementation is future work. We note that the deployment considerations discussed above are architectural guidelines derived from the benchmark-calibrated simulation environment and do not constitute validation on a physical cloud robotics system. Before deployment on a target platform, the calibration procedure described in Section 6.1 must be repeated to instantiate platform-specific task demands, node capacities, interference sensitivities, and energy parameters.

7. Conclusion and future work

This paper has studied interference-aware Makespan and energy scheduling for cloud robotics tasks. Rather than treating robotic workloads as generic computation jobs or addressing co-location interference in isolation, this work connects unified task profiling, benchmark-calibrated interference measurement, and allocation-order scheduling optimization into a single interference-aware optimization method, in which a unified quantitative profile of eight representative cloud robotics tasks is instantiated by lightweight tunable benchmarks to yield a pairwise interference sensitivity matrix, and an interference-aware formulation minimizing the weighted sum of Makespan and energy consumption is solved by the proposed IGA-CTS through a two-layer chromosome with model-coupled schedule evaluation. Within the benchmark-calibrated simulation environment, the experimental results demonstrate that interference-aware scheduling consistently improves upon its non-aware counterpart across all scales. Under the default balanced weighting, IGA-CTS provides a statistically supported and competitive overall Makespan–energy trade-off relative to the evaluated baselines, and the added weight-sensitivity analysis further shows that it retains the lowest normalized objective in most tested preferences.

Future work will extend the proposed method in six directions: (1) integrating deep learning with genetic algorithms to predict interference patterns and guide the search dynamically; (2) investigating Pareto-based selection, including non-dominated sorting, diversity preservation, and co-evolutionary mechanisms, to manage conflicting bi-objective search paths and adapt optimization preferences; (3) extending the method to hierarchical edge–cloud scheduling with dynamically arriving tasks and cross-domain coordination, allowing domain-level schedulers to operate in parallel; (4) incorporating deadline-aware mechanisms and deterministic continuum-level offloading for mixed-criticality workloads, thereby protecting deadline-critical tasks while retaining interference-aware Makespan–energy optimization for best-effort tasks; (5) incorporating robot–cloud transmission latency, bandwidth contention, and server-side co-location interference so that

Makespan captures both communication and computation delays; and (6) constructing and collecting dedicated ROS 2/Gazebo and physical robot-cloud traces containing task dependencies, multi-resource demands, co-location interference, and platform-level power and energy observations to validate the scheduler under realistic deployment conditions and recalibrate the interference and energy models.

CRedit authorship contribution statement

Chennian Xiong: Writing – original draft, Methodology, Conceptualization. **Weiwei Lin:** Writing – review & editing, Supervision. **Huikang Huang:** Validation. **Qi Mu:** Validation. **Keqin Li:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by Guangdong Provincial Natural Science Foundation Project (2025A1515010113), Shandong Provincial Natural Science Foundation Project (ZR2024LZH012), Key R&D Program of Shandong Province (2024CXPT066), and State Key Laboratory of Advanced Papermaking and Paper-based Materials (20262026TS02).

Data availability

Data will be made available on request.

References

- [1] X. Xie, X. Wu, Q. Hu, Bi-objective optimization for industrial robotics workflow resource allocation in an edge-cloud environment, *Appl. Sci.* 11 (21) (2021).
- [2] W.S. Wan, N.I.A. Apanidi, N.A. Muhammad, Task scheduling based on genetic algorithm for robotic system in manufacturing industry, *Int. J. Electr. Eng. Appl. Sci. (IJEEAS)* 5 (1) (2022).
- [3] S. Alirezazadeh, L.A. Alexandre, A survey on task allocation and scheduling in robotic network systems, *IEEE Internet Things J.* 12 (2) (2024) 1484–1508.
- [4] L. Khan, S.P. Isanaka, F. Liou, Fpga-based sensors for distributed digital manufacturing systems: A state-of-the-art review, *Sensors (Basel, Switzerland)* 24 (23) (2024) 7709.
- [5] C. Calderon-Cordova, R. Sarango, D. Castillo, V. Lakshminarayanan, A deep reinforcement learning framework for control of robotic manipulators in simulated environments, *IEEE Access* (2024).
- [6] C. Tang, S. Zhao, H. Su, B. Chen, A qos and sustainability-driven two-stage service composition method in cloud manufacturing: Combining clustering and bi-objective optimization, *J. Global Optim.* (2024) 1–36.
- [7] S. Alirezazadeh, L.A. Alexandre, Improving makespan in dynamic task scheduling for cloud robotic systems with time window constraints, *Clust. Comput.* 26 (3) (2023) 2027–2045.
- [8] G. Volpe, A.M. Mangini, M.P. Fanti, An architecture combining blockchain, docker and cloud storage for improving digital processes in cloud manufacturing, *IEEE Access* 10 (2022) 79141–79151.
- [9] L. Yin, J. Liu, F. Zhou, M. Gao, M. Li, Cost-based hierarchy genetic algorithm for service scheduling in robot cloud platform, *J. Cloud Comput.* 12 (1) (2023) 35.
- [10] H. Chen, Y. Wen, Y. Huang, C. Xiao, H. Dai, From integrated bridge system to marine bridge domain: A computational perspective, *Ocean Eng.* 298 (2024) 117171.
- [11] H. Etezadi, S. Eshkabilov, A comprehensive overview of control algorithms, sensors, actuators, and communication tools of autonomous all-terrain vehicles in agriculture, *Agriculture* 14 (2) (2024) 163.
- [12] Y. Wang, W. Tang, S. Xiong, A multi-task scheduling algorithm for cloud robots, in: 2019 IEEE International Conference on Service-Oriented System Engineering, SOSE, IEEE, 2019, pp. 344–3445.
- [13] L. Yin, J. Liu, Y. Fang, M. Gao, M. Li, F. Zhou, Two-stage hybrid genetic algorithm for robot cloud service selection, *J. Cloud Comput.* 12 (1) (2023) 95.
- [14] M. Sun, S. Quan, X. Wang, Z. Huang, Latency-aware scheduling for data-oriented service requests in collaborative iot-edge-cloud networks, *Future Gener. Comput. Syst.* 163 (2025) 107538.
- [15] A. Kouloumpis, G.L. Stavrinides, M.K. Michael, T. Theocharides, An optimization framework for task allocation in the edge/hub/cloud paradigm, *Future Gener. Comput. Syst.* 155 (2024) 354–366.
- [16] W. Lin, C. Xiong, W. Wu, F. Shi, K. Li, M. Xu, Performance interference of virtual machines: A survey, *ACM Comput. Surv.* 55 (12) (2023) 1–37.
- [17] M. Ruaro, H. Barral, M. Bertolino, R. Cataldo, R. Medina, M. Karaoui, E. Borde, The last-level-cache interference in guest performance: a case-study with zephyr os, in: 2023 26th Euromicro Conference on Digital System Design, DSD, IEEE, 2023, pp. 351–358.
- [18] B. Lu, J. Fang, X. Hong, J. Shi, Energy-efficient task scheduling for mobile edge computing with virtual machine i/o interference, *Future Gener. Comput. Syst.* 148 (2023) 538–549.
- [19] A. Tzenetopoulos, D. Masouros, S. Xydis, D. Soudris, Orchestration extensions for interference-and heterogeneity-aware placement for data-analytics, *Int. J. Parallel Program.* 52 (4) (2024) 298–323.
- [20] C. Delimitrou, C. Kozyrakis, ibench: Quantifying interference for datacenter applications, in: 2013 IEEE International Symposium on Workload Characterization, IISWC, IEEE, 2013, pp. 23–33.
- [21] Y.D. Barve, S. Shekhar, A. Chhokra, S. Khare, A. Bhattacharjee, Z. Kang, H. Sun, A. Gokhale, Fecbench: A holistic interference-aware approach for application performance modeling, in: 2019 IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2019, pp. 211–221.
- [22] R. Shaw, E. Howley, E. Barrett, An intelligent ensemble learning approach for energy efficient and interference aware dynamic virtual machine consolidation, *Simul. Model. Pr. Theory* 102 (2020) 101992.
- [23] S.M. Nabavinejad, M. Goudarzi, Data locality and vm interference aware mitigation of data skew in hadoop leveraging modern portfolio theory, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, 2018, pp. 175–182.
- [24] Z. Liu, Y. Cheng, C. Chen, J. Hu, R. Fu, D. Zhang, Isacpp: Interference-aware scheduling approach for deep learning training workloads based on co-location performance prediction, *IEEE Trans. Parallel Distrib. Syst.* 36 (8) (2025) 1591–1607.
- [25] S. Song, M. Xu, Z. Zhang, C. Gao, F. Zeng, Y. Ding, K. Ye, C. Xu, C-koordinator: Interference-aware management for large-scale and co-located microservice clusters, *Softw.: Pr. Exp.* 56 (5) (2026) 506–524.
- [26] A. Elgendy, J. Yan, M. Zhang, A parallel distributed genetic algorithm using apache spark for flexible scheduling of multitasks in a cloud manufacturing environment, *Int. J. Comput. Integr. Manuf.* 37 (5) (2024) 652–667.
- [27] G. Peng, Y. Wen, J. Liu, G. Kang, B. Zhang, M. Zhou, Energy-aware cloud manufacturing service selection and scheduling optimization, *Int. J. Comput. Integr. Manuf.* 38 (3) (2025) 309–334.
- [28] S. Liu, Q. Deng, X. Liu, Q. Luo, F. Li, C. Jiang, Dual-service integrated scheduling of manufacturing and logistics for multiple tasks in cloud manufacturing, *Expert Syst. Appl.* 235 (2024) 121129.
- [29] Q. Zhang, N. Li, J. Duan, J. Qin, Y. Zhou, Resource scheduling optimisation study considering both supply and demand sides of services under cloud manufacturing, *Systems* 12 (4) (2024) 133.
- [30] W. Xiong, Y. Wang, S. Gao, X. Huang, S. Wang, A multi-objective service composition optimization method considering multi-user benefit and adaptive resource partitioning in hybrid cloud manufacturing, *J. Ind. Inf. Integr.* 38 (2024) 100564.
- [31] Z. Chen, L. Zhang, X. Wang, K. Wang, Cloud-edge collaboration task scheduling in cloud manufacturing: An attention-based deep reinforcement learning approach, *Comput. Ind. Eng.* 177 (2023) 109053.
- [32] Y. Zhang, Z. Zhang, Y. Lu, H. Zhu, D. Tang, Dynamic decision-making for knowledge-enabled distributed resource configuration in cloud manufacturing considering stochastic order arrival, *Robot. Comput.-Integr. Manuf.* 87 (2024) 102712.
- [33] K. Li, Energy-constrained dag scheduling on edge and cloud servers with overlapped communication and computation, *J. Grid Comput.* 22 (3) (2024) 60.
- [34] K.M. Uma, S. Shukla, Energy and performance-aware workflow scheduler using dynamic virtual network resource optimization under edge-cloud platform, *Comput. Electr. Eng.* 123 (2025) 110085.
- [35] F. Liu, Z. Ma, B. Wang, W. Lin, A virtual machine consolidation algorithm based on ant colony system and extreme learning machine for cloud data center, *IEEE Access* 8 (2019) 53–67.
- [36] J. You, Z. Wu, W. Wei, N. Li, Y. Yang, Evolution of industrial robots from the perspective of the metaverse: Integration of virtual and physical realities and human-robot collaboration, *Appl. Sci.* 14 (14) (2024) 6369.
- [37] K. Chicaiza, R. Paredes, I.M. Sarzosa, S.G. Yoo, N. Zang, Smart farming technologies: A methodological overview and analysis, *IEEE Access* (2024).
- [38] H.K. Channi, Role of data acquisition, sensors, and actuators in mechatronics industry, *Comput. Intell. Tech. Mechatronics* (2024) 83–107.
- [39] A. Waqar, Intelligent decision support systems in construction engineering: An artificial intelligence and machine learning approaches, *Expert Syst. Appl.* 249 (2024) 123503.
- [40] D. Ma, B. Zhou, Data-driven multi-step nonlinear model predictive control for industrial heavy load hydraulic robot, 2024, arXiv preprint arXiv:2411.13859.

- [41] K. Aghababaiyan, B. Coll-Perales, J. Gozalvez, Scalable deterministic task offloading and resource allocation in the iot-edge-cloud continuum, in: 2025 IEEE Symposium on Computers and Communications, ISCC, IEEE, 2025, pp. 1–6, [Online]. Available: <http://dx.doi.org/10.1109/ISCC65549.2025.11326018>.
- [42] Q. Wang, D. Oswald, Confidential computing on heterogeneous cpu-gpu systems: Survey and future directions, *ACM Comput. Surv.* (2026).
- [43] K. Aghababaiyan, B. Coll-Perales, J. Gozalvez, Deterministic task offloading and resource allocation in the iot-edge-cloud continuum, in: 2025 IEEE 101st Vehicular Technology Conference (VTC2025-Spring), IEEE, 2025, pp. 1–6.
- [44] B. Alhijawi, A. Awajan, Genetic algorithms: Theory, genetic operators, solutions, and applications, *Evol. Intell.* 17 (3) (2024) 1245–1256.
- [45] M.I. Alghamdi, Optimization of load balancing and task scheduling in cloud computing environments using artificial neural networks-based binary particle swarm optimization (bpos), *Sustainability* 14 (19) (2022) 11982.
- [46] S.A. Kumer, N. Prabakaran, E. Mohan, B. Natarajan, G. Sambasivam, V.B. Tyagi, Enhancing cloud task scheduling with a robust security approach and optimized hybrid poa, *IEEE Access* 11 (2023) 122426–122445.
- [47] X. Zhang, A fine-grained task scheduling mechanism for digital economy services based on intelligent edge and cloud computing, *J. Cloud Comput.: Adv. Syst. Appl.* 12 (1) (2023) 30.
- [48] S. Katoch, S.S. Chauhan, V. Kumar, A review on genetic algorithm: Past, present, and future, *Multimedia Tools Appl.* 80 (5) (2021) 8091–8126.
- [49] W. Jing, Y. Zhang, Y. Chen, H. Zhang, W. Huang, Cloud-edge collaboration composition and scheduling for flexible manufacturing service with a multi-population co-evolutionary algorithm, *Robot. Comput.-Integr. Manuf.* 90 (2024) 102814.
- [50] E. Mustafa, J. Shuja, F. Rehman, A. Namoun, M. Bilal, A. Iqbal, Computation offloading in vehicular communications using ppo-based deep reinforcement learning: E. Mustafa and others, *J. Supercomput.* 81 (4) (2025) 547.
- [51] Y. Zhang, L. Tian, M. Guo, Am-a2c: Cloud computing task scheduling algorithm based on attention mechanism and action mutation, *Future Gener. Comput. Syst.* (2026) 108507.
- [52] A.P. Hadi, B. Hartono, K. Rozikin, I.S. Suasana, et al., Energy efficient task scheduling in serverless computing using pso and aco algorithms, in: 2025 4th International Conference on Creative Communication and Innovative Technology, ICCIT, IEEE, 2025, pp. 1–7.
- [53] U.K. Lilhore, S. Simaiya, Y.N. Prajapati, A.K. Rai, E.S. Ghith, M. Tlija, T. Lamoudan, A.A. Abdelhamid, A multi-objective approach to load balancing in cloud environments integrating aco and wwo techniques, *Sci. Rep.* 15 (1) (2025) 12036.



Chennian Xiong received the B.S. degree in computer science at South China University of Technology in 2017. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His research interests include cloud computing, performance interference modeling and interference-aware scheduling.



Weiwei Lin (Senior Member, IEEE) received the B.S. and M.S. degrees from Nanchang University in 2001 and 2004, respectively, and the Ph.D. in Computer Application from South China University of Technology in 2007. He has been a visiting scholar at Clemson University from 2016 to 2017. Currently, he is a professor in the School of Computer Science and Engineering at South China University of Technology. His research interests include distributed systems, cloud computing, and AI application technologies. He has published more than 200 papers in refereed journals and conference proceedings, such as ICML, NeurIPS, ACM



Huikang Huang received the MS degree in Computer Science and Theory from the South China Agricultural University, Guangzhou, China, in 2021, and the Ph.D. in Computer Science and Technology from South China University of Technology in 2026. His research interests mainly focus on cloud computing and energy efficiency optimization in distributed systems.



Qi Mu received his B.S. and M.S. degrees from Shandong University in 2013 and 2016, respectively, and is pursuing the Ph.D. degree with the School of Computer Science and Engineering, South China University of Technology, China. His research interests include Computer architecture, Heterogeneous acceleration, and Information security. He has published more than 10 papers in refereed journals and conference proceedings.



Keqin Li received a B.S. degree in computer science from Tsinghua University in 1985 and a Ph.D. degree in computer science from the University of Houston in 1990. He is a SUNY Distinguished Professor at the State University of New York and a National Distinguished Professor at Hunan University, China. He has authored or co-authored more than 1290 journal articles, book chapters, and refereed conference papers. He holds nearly 80 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top few most influential scientists in parallel and distributed computing, regarding single-year impact (ranked #2) and career-long impact (ranked #3) based on a composite indicator of the Scopus citation database. He is listed in Scilit Top Cited Scholars and is among the top 0.02% out of over 20 million scholars worldwide based on top-cited publications in the last ten years. He is listed in ScholarGPS Highly Ranked Scholars and is among the top 0.002% out of over 30 million scholars worldwide based on a composite score of three ranking metrics for research productivity, impact, and quality in the recent five years. He is a Member of Academia Europaea (MAE), a Member of the European Academy of Sciences and Arts (EASA), a Fellow of the American Association for the Advancement of Science (AAAS), a Member of the USA National Academy of Artificial Intelligence (NAAI), a Fellow of the International Academy of Artificial Intelligence Sciences (AAIS), a Fellow of the International Artificial Intelligence Industry Alliance (AIIA), a Member of the World Academy of Sciences (WAS), a Fellow of the World Academy of Engineering & Technology (WAET), a Fellow of the Institute of Electrical and Electronics Engineers (IEEE), a Fellow of the Asia Computational Intelligence Society (ACIS), and a Member of the SUNY Distinguished Academy.